Computationally Efficient Models for High-Dimensional and Large-Scale Classification Problems

Ma Li

School of Computer Engineering

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

2009

Abstract

Generally there are two main objectives in designing modern learning models when handling the problems with high-dimensional input spaces and a large amount of data. Firstly the model's *effectiveness* in terms of a good accuracy needs to be met and secondly the model's *efficiency* in terms of scalability and computation complexity needs to suffice. In practice these objectives require different types of learning models to solve different difficulties. In the case of the *parametric models* such as the radial basis function (RBF), the main difficulty is in the deterioration in accuracy and increase in computation complexity for high-dimensional data, which can be caused by the inductive nature of learning problems and the curse of dimensionality. While in the case of *nonparametric models* such as the Gaussian process (GP), the computing demand could become extremely high when there is a large amount of data to be processed. These difficulties pose the main obstacles preventing many successful traditional models from being applied to high-dimensional and large-scale data applications.

Several extensions have been proposed in recent years based on the traditional models, for both parametric and nonparametric types. Some of these extensions include the extended radial basis function (ENRBF) and the sparse pseudo-input Gaussian process (SPGP) models. With proper modifications, these models are both effective in improving the learning performance and efficient in reducing the computation complexity. However all of these models are meant to solve *regression* problems thus further extensions of these regression models are required to ensure that it can be applied to *classification problems*. Unfortunately such extensions are neither easy nor straightforward and careless extensions can bring about unpleasant results such as *masking* problems due to the different model assumptions and the introduction of complex computations introduced in classifications. Accordingly, the main goal of our research is to develop *effective classification* methods to improve the performance of such system while *efficiently* reduce the computation complexity with practical considerations.

In this thesis a thorough analysis of the extended regression models from both a theoretical and an experimental perspective will be presented. Two particular regression models, namely the parametric ENRBF model and the nonparametric SPGP model will be discussed in detail. Based on these discussions, both the theoretical foundations and the algorithm descriptions of the ENRBF Classifier (ENRBFC) and SPGP Classifier (SPGPC) will be discussed in detail. These proposed classification models further modify the corresponding regression models to ensure proper functionality for classification purpose. In addition, a series of properly designed experiments were carried out to compare the proposed models with some popular benchmarks. The potentials of these two algorithms are shown both theoretically and with experimental results from the benchmarked testing and real-world applications.

Acknowledgement

First of all I would like to express my special thanks to my supervisor Dr. Quek Hiok Chai and co-supervisor Dr. Abdul Wahab who have always been inspirational and enthusiastic in guiding my study. I am also very grateful to Dr. Ng Geok See for providing valuable advices and immense help with preparation of many of our papers.

I also wish to thank the Centre of Computational Intelligence (C2I) for providing a highly stimulating research environment. Many thanks go to the director Dr. Ong Yew Soon. A lot of my colleagues in the centre have been extremely receptive and supportive. For many useful discussions and their friendship, I would like to mention these people in particular: Dr. Tung Whye Loon, Dr. Liu Feng, Mr. Huang Haoming, and Mr. Wang Zhen.

Many people have helped me by providing professional knowledge and excellent suggestions at various times during my PhD, including Dr. Wlodzislaw Duch, Dr. Christopher Bishop, Dr. Carl Edward Rasmussen, Dr. Xu Lei and Dr. Sevki Erdogan.

For financial support I thank the Nanyang Graduate Scholarship and Dr. Lim Meng Hiot who has provided me the project officer position during my study.

Finally I would like to thank my family for their endless support, love and patience.

Contents

List of Figures

List of Tables

Chapter 1

1 Introduction

1.1 Learning from Data

In a typical scenario of *learning from data*, we are given a *set of training data*, in which both the outcomes (outputs) and features (inputs) of a set of instances can be observed. Depending on the applications we are modeling, the outcome measurements could be *quantitative* or *categorical*. Our task is to predict the outcomes for new unseen instances, based on the relationship between the outcome and features given in the training set. One way to solve this problem is to build a *learning model* to capture this relationship, and to make new predictions by evaluating this model on the new instances. The prediction accuracy is usually the most important criterion to measure the performance of these models. In *machine learning* [Mitchell 1997], the above problem is usually referred to as a *supervised learning* problem [Mitchell 1997; Cherkassky and Mulier 1998; Hastie, Tibshirani et al. 2001] because of the presence of outcome information in the learning process. Learning could also be *unsupervised*, where only the information about input features is available. We will focus on supervised learning in this thesis.

In today's science and industry, supervised learning has shown tremendous amount of importance in many applications such as:

- Estimating the risk of a disease, based on patients' clinical and demographic information.
- Predicting the Fraction of Inspired Oxygen (FiO2) amount in the ICU ventilation control, based on the time-series of FiO2 and other clinical variables.

- Predicting the localization site of E.Coli protein based on genetic sequence information.
- Identifying the human emotions based on the electroencephalography (EEG) signals collected from the subjects.
- Identifying the spam emails based on the frequency of several specific words or characters.
- Recognizing the digit numbers in a handwritten code, based on a digitized image.

Clearly, the above applications share some common characteristics that fit them into the supervised learning scenario discussed above. In each application, there is a set of input variables (e.g., clinical measurements, frequency of a word), and some desired output variables (e.g., FiO2 amount, classification of spam mails). More importantly, our commonsense knowledge makes us believe that there is a relationship between these inputs and outputs. By capturing this relationship into a learning model and evaluating it on new data, we can obtain useful predictions in these applications.

1.2 Main Challenges in Learning Problems

As discussed above, one of the biggest concern in *learning from data* is learning model's *prediction accuracy* for unseen data [Vapnik 1995; Hastie, Tibshirani et al. 2001; Bishop 2006]. This differentiates supervised learning problems from other data-driven problems such as traditional *pattern recognition* and *data mining* [Duda, Hart et al. 2001]. In the latter case the focus is usually more on finding patterns in existing data, and representing these patterns into effective knowledge forms. The generalization from observed training data to unseen essentially reveals the *inductive* nature of supervised learning [Vapnik 1995; Cherkassky and Mulier 1998; Bishop 2006]. Besides *prediction accuracy*, another important consideration in learning problems is the time complexity, namely the *training time* that a model is built based on training data and the *evaluation time* a model uses when making predictions for unseen data. In offline applications, the evaluation time is usually crucial, as learning models can be pre-trained before making predictions and extra time is usually allowed. In online (real-time) applications, however, both time complexities should be considered because the learning models is supposed to update with new instances and make predictions all on the fly. Bearing the accuracy criterion and the time complexities in mind, we will continue to discuss the three main challenges of the learning problems in the next section.

1.2.1 Inductive Nature of Learning

To better review the inductive nature of the learning problems, we put it into a general context—the *function approximation* problem [Hastie, Tibshirani et al. 2001]. Consider a training data set that consists of a set of instances $\{\mathbf{x}_i, y_i\}$, where $\mathbf{x}$ and y are input and output variables respectively. Assuming that a function $y(\mathbf{x})$ underlies the observed data, the task of function approximation is to infer the function from the given data, and predict its value at new input variables. In the literature, there are two main alternatives in estimating $y(\mathbf{x})$, namely *parametric* and *nonparametric* methods [MacKay 1998]. In a parametric approach we express the unknown function $y(\mathbf{x})$ in terms of a linear/nonlinear function $y(\mathbf{x};\boldsymbol{\theta})$ parameterized by parameters $\boldsymbol{\theta}$. Representatives of this type include multi-layer neural networks and radial basis functions (RBF). On the other hand, in nonparametric methods, predictions are obtained without giving the unknown function $y(\mathbf{x})$ an explicit parameterization; $y(\mathbf{x})$ lives in the infinite-dimensional space of all continuous functions of $\mathbf{x}$. A well known nonparametric approach is Gaussian process (GP), where without parameterizing $y(\mathbf{x})$, a prior $p\left(y\left(\mathbf{x}\right)\right)$ is directly placed on the space of functions. More detail will be covered in the following chapters. To simplify the discussion in this section, we take the parametric approaches as an example in the following.

In practice, some noise or artifacts could be present in both input and output variables. However, we are usually more interested in modeling this uncertainty in outputs as the final goal is to predict the output values for unseen data. Given a parameter vector $\boldsymbol{\theta}$, the learning problem discussed in the last section is in fact an approximation problem of the stochastic function $p(y\,|\,\mathbf{x},\boldsymbol{\theta})$ where $p(y\,|\,\mathbf{x})$ is essentially the conditional probability of output variable. Given a new input $\hat{\mathbf{x}}$, the knowledge about $p(y\,|\,\mathbf{x},\boldsymbol{\theta})$ will be enough to make prediction for new output $\hat{y}$, although different decision-making criteria of choosing different $\hat{y}$ exist in practice [Hastie, Tibshirani et al. 2001; Bishop 2006].

The approximation of $p(y\,|\,\mathbf{x},\boldsymbol{\theta})$ can be performed in several ways. Here we furhter assume that output y is quantitative, which corresponds to a regression task (see Section 1.3 for details). The discussion applies to categorical outputs as well, except for some technical differences that we will cover in Section 1.3. Let’s consider an approximation of $p(y\,|\,\mathbf{x},\boldsymbol{\theta})$ where the quantitative output y is assumed to be corrupted by an additive Gaussian noise, i.e., $y_i = f(x_i) + \varepsilon_i$.We further assume the noise is *white*, which means the noise ε_i follows a standard Gaussian distribution $p(\varepsilon) = \mathrm{N}(0,\sigma^2)$. The rationality of this assumption is supported by the *Central Limit Theorem*. Taking all these assumptions into account, the conditional probability function to be approximated can be expressed as:

$$p(y\,|\,\mathbf{x}) = \mathrm{N}\left(y\,|\,f(\mathbf{x}\,|\,\boldsymbol{\theta}),\sigma^2\right) \tag{1.1}$$

where $f(\mathbf{x}\,|\,\boldsymbol{\theta})$ is the *mean function* of output and usually referred to as *regression function* in the literature [Ripley 1996; Hastie, Tibshirani et al. 2001; Bishop 2006]. The mean function of $f(\mathbf{x}\,|\,\boldsymbol{\theta})$ is *deterministic* although the output's conditional probability function $p(y\,|\,\mathbf{x})$ is stochastic. Based on equation (1.1), the learning problem thus corresponds to the approximation of the parameter value $\boldsymbol{\theta}$. Estimating parameter values based on observations is a well-known problem in statistics. One popular method is to use *maximum likelihood (ML) estimation* [Hastie, Tibshirani et al. 2001; Bishop 2006]. The *likelihood* is the *log-sum* of the conditional probabilities of the training instances, as be described in equation (1.2)

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p\left(y_i \,|\, \boldsymbol{\theta}, \mathbf{x}_i\right) \tag{1.2}$$

Substituting equation (1.1) into (1.2), the likelihood can be further expanded as:

$$L(\boldsymbol{\theta}) = -\frac{N}{2}\log(2\pi) - N\log\sigma - \frac{1}{2\sigma^2}\sum_{i=1}^{N}\left(y_i - f\left(\mathbf{x}_i \,|\, \boldsymbol{\theta}\right)\right)^2 \tag{1.3}$$

By eliminating the $\boldsymbol{\theta}$-independent constants π and σ and taking the negative at both sides, maximizing (1.3) is equivalent to minimizing the following error term in equation (1.4).

$$RSS(\boldsymbol{\theta}) = \sum_{i=1}^{N}\left(y_i - f\left(\mathbf{x}_i \,|\, \boldsymbol{\theta}\right)\right)^2 \tag{1.4}$$

In the literature, equation (1.4) is usually referred to as *residual sum-of-squares* (RSS) error [Hastie, Tibshirani et al. 2001], which is one of several common error measurements for regression problems. By deriving equation (1.4) directly from equation (1.3) and (1.2), we can see that RSS is actually based on the ML estimation of the training data. Without other constraints, it is well known that minimization of (1.4) leads to infinitely many solutions—any function $f(\mathbf{x}\,|\,\boldsymbol{\theta})$ that passes through the training points $(\mathbf{x}_i, y_i)$ will satisfy as a solution, because $RSS(\boldsymbol{\theta})$ is reduced to exactly 0 in these cases. Even worse, any of those solutions could make poor predictions on new instances, because of the existence of noise in the training observations. In other words, the learning problem has an *inductive* nature, as generalizing from a finite number of instances to infinite unseen is inductive. This property of learning requires us to figure out a certain method that can help us pick one particular model from an infinite number of candidates.

In the literature several approaches have been proposed to solve this difficulty. Classical *model selection* methods [Hastie, Tibshirani et al. 2001; Bishop 2006] usually combines the direct minimization of RSS in equation (1.4) with some other criteria, which leads to a constrained minimization problem. For example, in the *regularization* methods such as *ridge regression* [Hastie, Tibshirani et al. 2001] (also known as the *weight decay* [Bishop 1995] in the neural network literature) and LASSO [Tibshirani 1996], different L_n *norms* are used as the additive constraints (or regularization terms) to be combined with the minimization of RSS. Minimizing those constrained criteria essentially corresponds to restricting the inner complexities of learning models [Cherkassky and Mulier 1998; Hastie, Tibshirani et al. 2001]. In addition to adding explicit constraints to RSS, we can also pick the appropriate model based on an averaged prediction error on all unseen data. Although the knowledge about the distribution of unseen data is lacked, the averaged prediction error can be approximated in a statistical way. A typical example of these methods is *cross-validation* [Hastie, Tibshirani et al. 2001] where the input training data is split and reused in a particular way. This method directly estimates the prediction error and chooses the appropriate model based on that estimation. Such model selection methods are widely used with traditional learning models.

Recently, some *Bayesian-learning* based methods [MacKay 2003; Bishop 2006; Rasmussen and Williams 2006] have been proposed, as an alternative to the classical *model selection* methods. Based on the above discussion, an important reason that forces us to pick a particular model from all candidates is that traditional learning methods usually make predictions for unseen by evaluating on one single model. Different from traditional methods, the Bayesian learning framework avoids this trouble as it is able to make predictions by considering all candidate models together. At the first glance it seems to be impossible to consider potentially infinite number of candidates. However, it turns out that by choosing some appropriate conjugate priors, this kind of calculation can be accomplished by the *marginalization* property of a certain stochastic process (e.g., the Gaussian process [Williams and Rasmussen 1996]). Bayesian methods usually bear close connections to *kernel* methods [Scholkopf and Smola 2002], because there is a distinct correspondence between covariance function of Bayesian methods and kernel functions.

1.2.2 Curse of Dimensionality

Besides the difficulties caused by the inductive nature, another big challenge in learning problem is the *curse of dimensionality* [Bellman 1961; Hastie, Tibshirani et al. 2001; Bishop 2006]. This problem has many different manifestations in the general context of learning in high dimensional space.

The first manifestation is that the complexity of the underlying function $f\left(\mathbf{x}\,|\,\boldsymbol{\theta}\right)$ in equation (1.4) usually grows rapidly with the number of input dimension. For example, assume the underlying function is a second-order polynomial. In one dimensional space, the number of coefficients to be estimated is 3; whereas when dimension becomes 5, this number increases to 10. And when the input dimension becomes 10, there will be a total number of 66 coefficients to be determined. Thus, there is a power law growth of the number of parameters in this case. As a consequence, if we want to get accurate estimations of parameters in a high dimensional space, the number of training data needs also to increase accordingly. This presents a big challenge in practice because the problems with tens or hundreds of input features are very common, and the underlying functions we are modeling are usually far more complicated than polynomials. In these cases the training data needed for estimations will be much more than what we can provide. At the first glance, it seems that some local non-parametric learning models (such as K-NN [Hastie and Tibshirani 1996; Hastie, Tibshirani et al. 2001]) could avoid this difficulty. Instead of estimating some explicit parameters, these non-parametric models make prediction on a new point $\hat{x}$ based on a *local neighborhood* around $\hat{x}$. For example, the prediction on $\hat{x}$ could simply be the mean value of the output observations in that neighborhood. From statistics theory, the accuracy of this prediction depends on whether we could find enough number of instances in a small neighborhood of $\hat{x}$. Let r denote the fraction of observations captured in $\hat{x}$'s local neighborhood. Thus the expected radius l of this neighborhood will be proportional to $r^{1/d}$, where d is input dimensionality [Hastie, Tibshirani et al. 2001]. To capture r=10% data in the neighborhood, the radius l will be 10% of the space range when d=1, and dramatically increase to 80% when d=10. In such a case the neighborhood will not be "local" anymore. In other words, the local non-parametric learning models also suffer from the curse of dimensionality.

Another manifestation of curse of dimensionality is that the distribution of the high-dimensional points is always close to the boundaries. To see this, suppose there are some data points that are uniformly distributed in a d dimensional hyper-sphere. The fraction of the data that are distributed in a thin shell right below the surface will be $1-(1-\varepsilon)^d$, where ε is the thickness of this shell [Bishop 2006]. Thus, in a ten-dimensional space, even when ε is a small value such as 0.1, this fraction will be about 65%. In other words, most of the high dimensional data are concentrated in a small sub-space that is very close to the boundaries. This will make the predictions for high dimensional data more difficult because now we need to extrapolate the neighbor points instead of interpolating them.

However, the difficulties caused by the curse of dimensionality do not prevent effective high-dimensional learning models from being found. The reasons are two-fold. Firstly, even in a high dimensional space, the data in practice usually exhibit some *smoothness* properties. When small changes occur on the input values, these smoothness properties make it impossible for the output values to change wildly without constraint. Thus, we can make reasonable assumptions on the local behavior of the learning models. For example, the model selection methods discussed in Section 1.2.1 could be used to control the model complexities and choose the optimal one from a set of candidates. Secondly, in practice high dimensional data are usually confined to a sub-space with a lower effective dimensionality. This usually happens when some input features are redundant or irrelevant to outputs, such that the effective number of input features is actually smaller than the number of features in the dataset. In addition, significant variations in the output could only occur along some particular directions in the original space. Thus, finding these effective low dimensional sub-spaces essentially reduces the dimensionality of the original problem. In Chapter 2 we will review some successful learning models that exploit one or both of the characteristics discussed above.

1.2.3 Finding Useful Features

As discussed in the previous section, a high dimensionality can be reduced by finding a lower-dimensional sub-space that captures the main variations of input-output changes. In practice this usually corresponds to choosing a sub-set of the most significant features from all. Even for applications with moderate dimensionality, ranking and choosing the most important features can bring useful benefits. For example, as a result of ignoring non-significant features, the computation cost can be reduced in both training and evaluation stages. In some real-time applications such as online face detection, this reduction could be crucial. In addition, ranking input features by their relative significance could provide additional information that is useful to human interpretation. This information is also useful in finding relationship between different features. For example, in the study of the genetic expression of a specific disease, this technique can be used to find related genes among thousands or even more.

In the literature, the process of finding the important features is usually referred to as *feature selection*. This is one of the most important parts in building learning models. Feature selection can be performed either as a separate process or integrated with the training stage of learning models. When performed separately, the feature selection stage is usually done before or interleavingly with the main training stage. And it is meant to be relatively fast compared to the main training stage. Representatives in this category include unsupervised learning methods such as clustering, forward feature selection such as information gain or naïve Bayesian model [Bishop 2006], and the least absolute shrinkage and selection operator (LASSO) [Tibshirani 1996]. For example, by first clustering the data, we can find some prototypes in the data points and project the original data to the lower-space spanned by those prototypes. However in practice these methods usually have their limitations when handling complicated learning problems. For example, the results of clustering methods usually lack a direct connection to prediction performance as they ignore output data in finding prototypes. The naïve Bayesian based method depends on the assumption that the input features are independent of each other, which could not be the case in practice. The LASSO method can effectively improve the accuracy of feature selection results in terms of a better prediction on new data. But it depends on a linear regression model under the hood, which usually limits its flexibility in dealing with complicated learning tasks.

Recently there are some new learning models proposed that can do feature selection and learning/evaluation as a whole process. As a result, these models usually exploit the same criterion for both feature selection and model training, and thus can select important features directly based on learning accuracy. The cost is an increase in the computation complexity and in some cases only approximations to the full computation can be performed. The Gaussian process models [Rasmussen and Williams 2006] and its sparse approximations [Minka 2001; Smola and Bartlett 2001; Csato and Opper 2002; Lawrence, Seeger et al. 2003; Seeger, Williams et al. 2003; Candela and Rasmussen 2005] are examples of this category and will be discussed in detail in Chapter 2.

1.3 Regression and Classification

According to the nature of output variables, supervised learning problems can be further divided into *regression* and *classification* problems. Their difference lies on whether the output values are *quantitative* (in regression) or *qualitative* (in classification). The example of the Fraction of Inspired Oxygen (FiO2) prediction in Section 1.1 is a regression problem, as its outputs are quantitative values. On the other hand, the digit image recognition problem is a classification problem, where the output values are a set of class labels from 0 to 9.

Although these labels are also numbers, there is no explicit ordering among them, which is an important characteristic of qualitative variables.

Regression and classification problems also bear close connections to each other. In fact, both of them can be viewed as special cases of *function approximation* problems in a general context. Since the output values of regression are quantitative, the corresponding functional approximation is straightforward. The function to be approximated in regression problems is the one mapping the input to the noise-free output, also known as the *regression function* in the literature. In the classification, however, there are some constraints on their output values, since they are essentially a set of class labels. To apply function approximation techniques, the classification outputs are usually *encoded* in some way. For example, in a binary classification problem such as the spam/non-spam mail recognition, the output can be encoded as a two-value variable with values 0/1 or -1/+1. In multi-classifications, however, the *dummy-variable* encoding scheme is widely used [Hastie, Tibshirani et al. 2001]. Generally, this scheme uses a k-length bit vector to represent a k-class output. Only one bit is turned on to encode a specific class. Taking the digit image recognition problem as an example, the output for “digit 0” can be encoded as $(1,0,0,0,0,0,0,0,0,0)$, and “digit 2” as $(0,0,1,0,0,0,0,0,0,0)$. This scheme may seem a bit redundant at the first glance but it enforces symmetry of the class labels and thus facilitates many classification models. Thus, the function approximation problem in this scenario corresponds to approximating the function that maps the inputs to one of the dummy-variable representations.

On the other hand, the differences between the two types of problems are even bigger than their similarities [Bishop 2006; Rasmussen and Williams 2006]. These differences are the main obstacles that prevent regression models from being directly applied to classification problems. In Section 1.2.1, we modeled the conditional probability $p(y \mid \mathbf{x})$ for regression problems as a Gaussian distribution (as shown by equation (1.1)). The rationality of this assumption is directly based on the *continuous* nature of the output values. However, in classification problems this assumption is not appropriate anymore because the output values (in this case the class labels) in classification are *discrete*. Accordingly a more appropriate model of $p(y \mid \mathbf{x})$ would be a multinomial distribution. This difference heavily increases the computation complexity for classification problems, because now the logarithm of the probabilities in the computation of likelihood will require a more complicated representation than those derived from Gaussian. As a result there will be lots of nonlinear approximations involved in the calculation of these quantities.

1.4 Motivations

So far we have discussed the main challenges in learning problems and the distinctions between regression and classification tasks. In the recent years, there have been a lot of studies focusing on the developments of effective and efficient learning models. Compared to traditional models, these models are able to provide accurate predictions in a relatively fast learning/evaluating process. More importantly, they are able to scale with the increased complexity incurred by either more training instances or more input features. By exploiting some general strategies, these models achieve a good balance between prediction accuracy the computation complexity, and thus effectively solve those learning challenges in an efficient manner. Among these models of particular interest are two: the *Radial Basis Function* (RBF) model [Powell 1987] and the *Gaussian Process* (GP) model [Williams and Rasmussen 1996]. As the respective representations of non-parametric and parametric models, both models can achieve good performances on small-scale and low-dimensional data. And to handle the situation where more instances or input features are involved, both models have been recently extended. The details of the extension work will be discussed in the following chapters.

However, these extended RBF and GP models are originally proposed to deal with regression problems and the further extension to classification is not trivial. Thus, in this thesis, the powers of these models are enhanced to handle the classification problems. To summarize, the objectives of this research are:

1. To develop classification models that are able to *scale* with more input features and more training instances. The developed models should be resilient to avoid any significant deterioration of learning accuracy and computation cost, which is usually suffered by some traditional models when more input variables or training instances are involved.

2. To provide useful *feature selection* results along with the training process of the developed classification models. These results enable us to rank the relative importance of different features and provide intuitive explanations to the models.

The experimental results show a promising potential of the proposed models in many real-world applications, especially the medical data classification problems where tens or hundreds of input features are usually involved and the explanations of learning results are as important as a good learning accuracy. In these applications, a lot of models suffer from significant performance deterioration caused by the curse of dimensionality and intuitive explanations to learning results are usually lacked. These difficulties impair the usefulness of such models and could eventually prevent them from being used in practice. On the other hand, the models proposed in this thesis focus on how to effectively and efficiently handle these difficulties by avoiding accuracy deterioration and providing statistics-based results and useful feature rankings.

1.5 Thesis Organization

Following the introduction in this chapter, Chapter 2 continues to review the two main learning paradigms in the literature, namely the *parametric* and *nonparametric* learning models. Based on the general discussion, we also introduce a specific parametric model—the traditional RBF model and one of its extension ENRBF in that chapter. These two models form the basis of our own research work that is introduced in Chapter 3—the ENRBFC model, which is an extension of ENRBF model for classification problems. In Chapter 4, we turn our attention to nonparametric models and specifically introduce the GP model as one of the representatives. We discuss both the advantages and limitations of the traditional full GP model and its possible extensions when handling large scale problems. In Chapter 5, we explore the possibility of extending a special GP model, namely SPGP, to handle classification problems. The main contributions of this thesis are the two new classification models proposed in Chapter 3 and 5. In Chapter 6, we conclude the whole thesis and also discuss several directions for future work.

Chapter 2

2 A Review of RBF and ENRBF Model

In this chapter, we start with an introduction of the two main paradigms for learning problems, namely the parametric and nonparametric learning. After that, a specific parametric model, the Radial Basis Function (RBF) model [Powell 1987; Hastie, Tibshirani et al. 2001] and one of its extension, the Extended Normalized RBF (ENRBF) model [Xu 1998] are introduced in detail. These two models form the basis our discussion in Chapter 3.

2.1 Two Main Learning Paradigms

Generally, there are four main stages involved in building a learning model, namely the *training stage* (optimizing model parameters), the *model selection stage* (preventing over-fitting by selecting appropriate model complexity), the optional *feature selection stage* (selecting the most relevant features) and the *evaluation stage* (making predictions on new data). Based on the different strategies used at different stages, there are two main paradigms of learning models in the literature—the *parametric* and *non-parametric* models [Cherkassky and Mulier 1998; MacKay 1998; Hastie, Tibshirani et al. 2001]. In a parametric model, there are some explicit parameters that are able to determine the model's behavior. After the training stage, these parameters are optimized and will provide enough information for any future evaluations on new data. For example, a linear model can be fully determined by its coefficients. After the coefficients are optimized, the training data can be safely discarded. Most of the traditional models such as Generalized Additive Models (GAM) [Hastie and Tibshirani 1990], Cerebellar Associative Memory Approach (CMAC) [Nguyen, Shi et al. 2006; Teddy, Lai et al. 2008; Teddy, Quek et al. 2008], Neural Networks (NN) [Bishop 1995], Neural Fuzzy System (NFS) [Harris, Hong et al. 2002; Nguyen and Shi 2008; Singh, Quek et al. 2008], Radial Basis Functions (RBF) [Powell 1987] and Relevance Vector Machine (RVM) [Tipping 2001] belong to this category. On the other hand, there are no explicit parameters in non-parametric models. In fact, these models' evaluations on new data are based on both the training data and the new instances to be evaluated. However, there could be some explicit hyper-parameters in these nonparametric models. Unlike model parameters that directly determine the evaluation values of models, hyper-parameters essentially control the complexity of models. For example, in the *k* nearest neighbors (*k-NN*) method [Hastie, Tibshirani et al. 2001], the parameter *k* determines the neighborhood size of its local estimation, and is usually used to control the model's complexity/flexibility. As a result, the parameter *k* is usually determined by the model selection stage. After *k*'s value is determined, the model's evaluation values will only depend on the training data and the instances to be evaluated. Typical representatives of non-parametric models are the *kernel* based methods [Scholkopf and Smola 2002] and Gaussian Process (GP) [Rasmussen and Williams 2006]. In the following discussions, we will compare these two paradigms in detail by discussing their different learning stages.

2.1.1 Parametric Models

The *training stage* of parametric models corresponds to the optimization of an explicit model parameter vector $\boldsymbol{\theta}$. After that, the training data can be discarded and the predictions for new instances are solely based on these parameters. From a Bayesian point of view [MacKay 2003; Bishop 2006], the optimization of $\boldsymbol{\theta}$ essentially corresponds to estimating its *posterior probability* given the training data as observations, i.e., $p\left(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{y}\right)$, where $\mathbf{X}$ and $\mathbf{y}$ are training inputs and outputs respectively. This viewpoint is actually general enough to cover both traditional and Bayesian parametric models. In a traditional parametric model such as

RBF, after $p(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{y})$ is derived, a further step is taken to get a point estimate of $\boldsymbol{\theta}$. This point estimate is usually the *mode* of $p(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{y})$ which corresponds to the value with the biggest probability. On the other hand, the Bayesian parametric models such as RVM are able to evaluate new instances directly based on $p(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{y})$.

One simple measure of the complexity of a parametric model is the number of components in vector $\boldsymbol{\theta}$, i.e., the number of scalar parameters it comprises. A complex and flexible model usually involves more parameters than a simple one. The appropriate number of parameters for a type of models can be determined by a model selection method such as *k*-fold CV. However, a more elegant way of controlling the model complexities is to use *penalized regularization*. For example, for a regression problem, a penalized learning criterion consists of the ordinary *residual sum-of-squares* (RSS) and a penalty term:

$$PRSS(f;\lambda) = RSS(f) + \lambda J(f) \tag{2.1}$$

Generally, the penalty term $J(f)$ should measure the *roughness* of a model f, i.e., how rapidly f varies in small regions of input space, and λ is its weight coefficient. A popular choice of $J(f)$ is the l_n norm of the parameters $\|\boldsymbol{\theta}\|^n$. This method is usually dubbed as *parameter shrinkage* in the literature. When *n*=2, it is also called *ridge regression* [Hastie, Tibshirani et al. 2001] or *weight decay* [Bishop 1995]. The special case of *n*=1 is known as the LASSO [Tibshirani 1996] in the statistics literature. LASSO is of great interests for its ability to generate *sparse* models—when λ is sufficiently large, some of the parameters θ_j are driven to zero. This leads to a sparse model in which the corresponding features/basis functions play no role. As discussed in Chapter 1, a sparse model's result can be used to alleviate the curse of dimensionality and find useful features.

Because regularization/shrinkage methods essentially limit a model's effective complexity, they make it possible to train very flexible models on datasets of limited size and avoid over-fitting. At the model selection stage, the problem of determining the optimal model complexity is equivalent to determining a suitable value of the regularization coefficient λ. When $\lambda = 0$, there is actually no penalty on the model f, and any function that passes through the training points will minimize the RSS term to exactly 0. On the other side, setting $\lambda = \infty$ will put extremely strong constraints on f, which only leaves a constant-value model as a possibility. In practice, depending on the inner complexity of data to be modeled, the optimal λ should be a value between the two extreme cases. In practice, the optimal value of λ is usually estimated at the model selection stage, by methods such as cross validation.

In the literature parametric models are also further divided into *linear* and *nonlinear models*, depending on whether a model's output $y(\mathbf{x};\boldsymbol{\theta})$ can be expressed as a linear form of the parameters $\boldsymbol{\theta}$ (see Section 1.2.1 for the definition of parametric models). For example, as a linear parametric model, the Generalized Additive Model (GAM) can be expressed as a linear sum of several fixed basis functions. During learning process these basis functions are fixed and only their coefficients are tunable. In contrast, in nonlinear parametric models the basis functions are also parameterized and tunable during learning. One representative of nonlinear parametric models is the Multi-Layer Perception model (or Neural Network) [Bishop 1995].

2.1.2 Non-parametric Models

Non-parametric models are essentially *memory-based* methods in the sense that instead of estimating and evaluating on some explicit learning parameters, they make predictions for new observations directly based on training data. Thus, the training data must be kept even after the learning stages. On the other hand, this nonparametric nature also makes these models more flexible than parametric models because their effective complexities are able to automatically increase with the size of training data set.

As typical representatives of nonparametric models, the kernel based methods usually use some *similarity measure* to build associations of different instances in the input space. These measurements are usually referred to as *kernel functions* [Rasmussen and Williams 2006] in the literature, which are analogous to the *inner products* of input instances. However, kernel functions are different from inner products in that they are not necessarily defined in the original input space. Intuitively, a lot of kernel functions can be viewed as transforming the original vectors to another space (usually with higher or even infinite dimensions) and taking the normal inner products in that transformed space. For nonparametric models, the calculations of kernel functions are the major computation costs for both training and evaluation stages. In addition, a lot of kernel functions $k(\mathbf{x},\mathbf{x}')$ have a good property that their computation overheads are usually independent of input dimensionality. At the first glance, this seems to be able to escape the curse of dimensionality. But it turns out that the computing overhead is now shifted to the computation of the kernels for every pair in the training set. In other words, the computation cost of kernel methods depend heavily on the size of training data set. In section 4.1 we will see that this presents the major problem that limits the usage of many kernel methods for large data set.

To select an appropriate complexity for nonparametric models, the regularization method used for parametric models (Equation (2.1)) need to be generalized because there are no explicit parameter vectors in this case. One way for this generalization is to from the Bayesian perspective [Neal 1996] of model selection criteria. From this viewpoint, the model selection criteria (such as the PRSS in equation (2.1)) essentially represent the *posterior probability* of a specific model $p(f \mid \mathbf{X}, \mathbf{y})$, where f is the learning model and $\mathbf{X}$ and $\mathbf{y}$ are the training inputs and outputs. Based on Bayesian theorem, the logarithm of $p(f \mid \mathbf{X}, \mathbf{y})$ can be expressed as,

$$\begin{aligned} \log p(f \mid \mathbf{X}, \mathbf{y}) &= \log\left[p(\mathbf{y} \mid \mathbf{X}, f) p(f \mid \mathbf{X}) \right] \\ &= \log p(\mathbf{y} \mid \mathbf{X}, f) + \log p(f \mid \mathbf{X}) \\ &= \log p(\mathbf{y} \mid \mathbf{X}, f) + \log p(f) \end{aligned} \tag{2.2}$$

In the last step of above equation we have $\log p(f \mid \mathbf{X}) = \log p(f)$ because the learning model f only models the outputs $\mathbf{y}$ and is independent of inputs $\mathbf{X}$. On the right side of the equation is the sum of two terms, where $\log p(\mathbf{y} \mid \mathbf{X}, f)$ is the logarithm of the *likelihood* of training data, and $\log p(f)$ is the logarithm of the model's *prior probability*. When compared to each other, the connection between equations (2.1) and (2.2) are obvious—the data-related term RSS corresponds to the likelihood, as we have introduced in Section 1.2.1, and the penalty term $\lambda J(f)$ just corresponds to the model's log prior probability. Thus the regularization method still applies to the nonparametric methods if we can work out the corresponding probability functions. The details of different choices will be covered in the following sections. Although a point estimate could be made by choosing the mode of $p(f \mid \mathbf{X}, \mathbf{y})$, a lot of nonparametric models (especially the Bayesian kernel methods) make new predictions by directly using $p(f \mid \mathbf{X}, \mathbf{y})$ through *marginalization*.

2.1.3 Comparisons of the Two Paradigms

Both parametric and nonparametric models have had a lot of successful applications. As a special type of parametric models, the linear parametric models are easy to compute, since their determinations of parameters only involve *linear* optimizations. In some cases when the training data size is relatively small, a linear model could be the only choice that will not cause an over-fitting. However, these models could also suffer from the simplicity of their linear assumption, since in practice there are a lot of data bearing more complicated patterns than a linear relationship. On the other hand, the nonlinear parametric models, such as MLP and RBF, allow their basis functions to adapt as well. This greatly increases the flexibility and modeling ability of the models. In fact, a lot of adaptive parametric models are *universal approximators* in the sense that they can approximate arbitrarily complex data given enough number of parameters are used. But this increased flexibility comes at a cost—the optimization of parameters in these models is now nonlinear, which introduces a lot of complexities into the computing. Nonparametric models solve this flexibility-simplicity dilemma from another direction. They lift the limitation of fixed basis functions by using enough of them, i.e., typically infinitely many, and over-fitting problem can be controlled by using appropriate priors or regularization methods [Rasmussen and Williams 2006]. Compared to nonlinear parametric models, nonparametric models are usually much easier to implement in practice and have the same flexibility. However, a major difficulty of nonparametric models is their high computation cost for when the number of instances is large. The details about how to handle these difficulties are the focus of this thesis and will be discussed thoroughly in the following chapters.

2.2 RBF Model and EM Learning Method

In the following sections, we will introduce a special parametric model—the RBF model and its extension the ENRBF model in detail. The discussions here form the basis of the research work in the next chapter.

As a special case of parametric models, the *Radial Basis Function* (RBF) model has been studied and widely used since its proposal in [Powell 1987]. Similar to many parametric models, it has the universal approximating ability for an arbitrary complicated function. However, compared to other models such as *Multilayered Neural Networks*, one of the advantages of RBF is its relatively fast *two-step training method*. At its training stage, two separate steps are performed to determine the parameters of its basis functions and the corresponding basis coefficients respectively. In detail, the parameters of the basis functions are usually estimated by some fast and unsupervised training methods, such as *clustering* methods [Jain, Murty et al. 1999]. After that, determining the coefficients of the basis functions can be simplified as solving a linear system (e.g., by the Moore-Penrose pseudo-inverse), which is also very fast. Compared to the nonlinear optimization methods that are exploited by some models such as neural networks, this two-step method is usually much faster and easier to implement in practice.

However, the drawback of the two-step training is also obvious—the accuracy of the learning models could deteriorate due to the unsupervised nature of determining the parameters of the basis functions. Recently a new training method for RBF that is based on the *Expectation-Maximization* (EM) algorithm [Dempster, Laird et al. 1977] has been proposed to overcome this weakness [Ma, Ji et al. 1997; Ma and Ji 1998; Orr 1998; Xu 1998; Marcelino, Ignacio et al. 2003]. Compared to the two-step method, the EM based method is able to significantly improve the accuracy, and remain fast and simple in the mean time. The structure of RBF, its training methods and its main difficulties are reviewed in the following sections.

2.2.1 RBF Model and Its Traditional Learning

Following [Xu 1998] and [Xu, Jordan et al. 1995], an RBF model can be viewed as a linear combination of several basis functions,

$$f(\mathbf{x}) = \sum_{j=1}^{M} w_j \phi_j\left(\mathbf{x} \mid \boldsymbol{\theta}_j\right) \tag{2.3}$$

where $\mathbf{x}$ is an input vector, ϕ_j is the *j*th basis function parameterized by $\boldsymbol{\theta}_j$, and w_j is the coefficient (or weight) of the *j*th basis function. Popular choices for the basis functions are *Gaussian* function, *Thin Plate Spline* function and etc. Most of these choices are *local* functions in the sense that the function values dramatically decrease to zero beyond a local region. Taking the Gaussian function as an example, the basis function $\phi_j\left(\mathbf{x} \mid \boldsymbol{\theta}_j\right)$ is chosen to be a Gaussian shaped function $\exp\left(-\frac{1}{2}\left(\mathbf{x}-\mathbf{c}_j\right)^T \boldsymbol{\Sigma}_j^{-1}\left(\mathbf{x}-\mathbf{c}_j\right)\right)$, where $\mathbf{c}_j$ is the center of a local neighborhood and the covariance matrix $\boldsymbol{\Sigma}_j$ actually measures the effective extent of the neighborhood. The only requirement for $\boldsymbol{\Sigma}_j$ is to be symmetric and positive semi-definite. By specifying different types of covariance matrices such as diagonal or isotropic, the shapes of the neighborhood could be different. Although the basis functions could also be selected as global functions such as the polynomial functions, they are rarely used in practice. Compared to the local basis functions, these global functions have the risk of flapping about madly in remote regions, when tweaking the coefficients to achieve functional form in one region [Bishop 2006].

The RBF model can also be recast to a normalized form, resulting in the *Normalized Radial Basis Function* (NRBF) model [Bugmann 1998],

$$f(\mathbf{x}) = \sum_{j=1}^{M} w_j \tilde{\phi}_j \left(\mathbf{x} \mid \boldsymbol{\theta}_j\right) \tag{2.4}$$

Here the normalized basis function $\tilde{\phi}_j\left(\mathbf{x} \mid \boldsymbol{\theta}_j\right) = \phi_j\left(\mathbf{x} \mid \boldsymbol{\theta}_j\right) \Big/ \sum_{j'=1}^{M} \phi_{j'}\left(\mathbf{x} \mid \boldsymbol{\theta}_{j'}\right)$. Compared to the standard RBF model, NRBF has some computation advantages under some circumstances [Bugmann 1998]. Moreover, NRBF model has a direct connection to the *Mixture-of-Expert* model [Jordan and Jacobs 1994; Xu 1998], which directly implies the EM based training method for NRBF/RBF. Details about this training algorithm will be discussed in Section 2.2.2.

In regression cases, RBF models can be trained by minimizing an RSS (in equation (1.4)) or PRSS criterion (in equation (2.1)). Recall that $RSS = \sum_{i=1}^{N}\left(y_i - f\left(\mathbf{x}_i\right)\right)^2$ and $PRSS = RSS\left(f\right) + \lambda \sum_{j=1}^{M} \| w_j \|^p$, where $\mathbf{x}_i$ and y_i are training inputs and outputs. In the two-step training algorithm, some fast unsupervised algorithm is first used to estimate the parameter $\boldsymbol{\theta}_j$ of each basis function. For example, $\boldsymbol{\theta}_j$ could be estimated by fitting a *mixture of Gaussian* distribution to the training inputs through some clustering methods. After that, the basis functions $\phi_j\left(\mathbf{x} \mid \boldsymbol{\theta}_j\right)$ or $\tilde{\phi}_j\left(\mathbf{x} \mid \boldsymbol{\theta}_j\right)$ can be viewed as fixed, and the problem of determining basis coefficients w_j is converted to solving a linear system $\mathbf{y} = \boldsymbol{\Phi}\mathbf{w}$. Here $\boldsymbol{\Phi}$ is the design matrix composed of the basis functions $\begin{pmatrix} \phi_1\left(x_1\right), \ldots, \phi_M\left(x_1\right) \\ \vdots \\ \phi_1\left(x_N\right), \ldots, \phi_M\left(x_N\right) \end{pmatrix}$ and $\mathbf{y}$ is the column vector of outputs $\left[y_1, \ldots, y_N\right]^T$. Directly minimizing RSS corresponds to finding the least square solution $\hat{\mathbf{w}} = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^T \mathbf{y}$. However, it could cause ill conditioning if the rank of $\boldsymbol{\Phi}$

is not full (e.g., when two basis functions are correlated). To stabilize the solution, the least-sum-squares and least-norm estimation $\hat{\mathbf{w}} = \left(\mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{I}\right)^{-1}\mathbf{\Phi}^T\mathbf{y}$ is usually used in practice, where $\mathbf{I}$ is the identity matrix and λ is the regularization parameter. In essence, this corresponds to minimizing the PRSS criterion with a l_2-norm penalty $\lambda\sum_{j=1}^{M}\| w_j \|^2$. The value of λ is usually chosen at the model selection stage as it essentially control the model complexity. Alternatively, the number of basis functions can also be used to control the complexity of an RBF model. Fitting a group of RBF models with varying number of basis functions, we can choose the appropriate one by cross validation method. In practice this method is usually preferred to the above regularization method due to its easier implementation.

As discussed at the beginning of this section, the two-step training method has a risk of impairing the learning accuracy due to the ignorance of output information. This can be overcome by the proposal of an EM-based learning algorithm [Xu 1998]. We will briefly review this training method in the next section.

2.2.2 EM Training Algorithm for RBF

As a general statistical method for parameter estimation, the *Expectation Maximization* algorithm (EM) [Dempster, Laird et al. 1977] has been widely used in different fields of the machine learning area [Ma, Ji et al. 1997; Gan and Harris 2001; Xu 2004]. Its rationality for training RBF models depends on an underlying connection between the RBF model and a more general statistical model—the *Mixture-of-Experts* (ME) model. We start this section with a brief introduction of the ME model.

In Section 1.2.1 we have discussed that there are often some uncertainties in training data. As a statistical model, the Mixture-of-Experts framework [Jordan and Jacobs 1994] directly models this uncertainty by a conditional probability:

$$p(y \mid \mathbf{x}) = \sum_{j=1}^{M} p(y \mid \mathbf{x}, j)\, p(j \mid \mathbf{x}) \tag{2.5}$$

where $p(j \mid \mathbf{x})$ is usually referred to as the jth *gating function* and $p(y \mid \mathbf{x}, j)$ is the jth *expert*. Based on the equation, the conditional probability $p\left(y \mid \mathbf{x}\right)$ is essentially represented by a mixture model of individual experts. The predictions for new instances can be made by deriving the *regression function* (discussed in Section 1.2.1) as,

$$\begin{aligned} f\left(\mathbf{x}\right) &= E\left[y \mid \mathbf{x}\right] \\ &= \int y \sum_{j=1}^{M} p(y \mid \mathbf{x}, j)\, p(j \mid \mathbf{x}) dy \\ &= \sum_{j=1}^{M} p(j \mid \mathbf{x}) \int y p(y \mid \mathbf{x}, j) dy \\ &= \sum_{j=1}^{M} \mu_j^y\, p(j \mid \mathbf{x}) \end{aligned} \tag{2.6}$$

where $p(j \mid \mathbf{x}) = p(\mathbf{x}, j) \Big/ \sum_{j'=1}^{M} p(\mathbf{x}, j')$ is actually the posterior probability of the jth expert generating instance $\mathbf{x}$, and $\mu_j^y = E\left[y \mid \mathbf{x}, j\right]$ is the conditional mean of the outputs of the jth expert.

The connection between NRBF and ME is quite obvious when we compare equation (2.6) with the NRBF model represented by equation (2.4). It turns out that the NRBF model can be viewed as a special case of ME, by viewing the basis coefficient w_j as μ_j^y and the normalized basis functions $\tilde{\phi}_j\left(\mathbf{x}\,|\,\boldsymbol{\theta}_j\right)$ as $p(j\,|\,\mathbf{x})$ for a specific *j*. If we further assume that the joint probability $p(\mathbf{x}, j)$ in ME is Gaussian, i.e.,

$$\begin{aligned} p(\mathbf{x}, j) &= p(j)\,p\left(\mathbf{x}\,|\,j,\boldsymbol{\theta}_j\right) \\ &= p(j)\,\mathrm{N}\left(\mathbf{x};\boldsymbol{\mu}_j^{\mathbf{x}},\boldsymbol{\Sigma}_j^{\mathbf{x}}\right) \end{aligned} \tag{2.7}$$

Then the basis functions of NRBF model will also have a Gaussian shape since $p(\mathbf{x}, j)$ essentially corresponds to the unnormalized basis functions.

Thus, the training of RBF models is equivalent to estimating the parameters $\left\{\mu_j^y,\boldsymbol{\mu}_j^{\mathbf{x}},\boldsymbol{\Sigma}_j^{\mathbf{x}}\right\}_{j=1}^M$ of ME models. In statistics, these parameters are usually optimized by the *maximum likelihood* (ML) estimation. In detail, we derived the joint likelihood of the training data based on a a Gaussian conditional *probability density function* (pdf) assumption,

$$p(y\,|\,\mathbf{x}, j) = \mathrm{N}\left(y;\mu_j^y,\left(\sigma_j^y\right)^2\right) \tag{2.8}$$

And by combining equations (2.7) and (2.8), the joint distribution can be represented by

$$\begin{aligned} p\left(\mathbf{x}, y\right) &= \sum_{i=1}^{M} p(j)\,p(\mathbf{x}, y\,|\,j) \\ &= \sum_{i=1}^{M} p(j)\,p(\mathbf{x}\,|\,j)\,p(y\,|\,\mathbf{x}, j) \\ &= \sum_{i=1}^{M} p(j)\,\mathrm{N}\left(\mathbf{x};\boldsymbol{\mu}_j^{\mathbf{x}},\boldsymbol{\Sigma}_j^{\mathbf{x}}\right)\mathrm{N}\left(y;\mu_j^y,\left(\sigma_j^y\right)^2\right) \\ &= \sum_{i=1}^{M} p(j)\,\mathrm{N}\left(\mathbf{x}, y;\boldsymbol{\mu}_j^{\mathbf{x},y},\boldsymbol{\Sigma}_j^{\mathbf{x},y}\right) \end{aligned} \tag{2.9}$$

where $\boldsymbol{\mu}_j^{\mathbf{x},y} = \begin{pmatrix} \boldsymbol{\mu}_j^{\mathbf{x}} \\ \mu_j^y \end{pmatrix}$ is the joint mean and $\boldsymbol{\Sigma}_j^{\mathbf{x},y} = \begin{pmatrix} \boldsymbol{\Sigma}_j^{\mathbf{x}} & \mathbf{0} \\ \mathbf{0}^T & \left(\sigma_j^y\right)^2 \end{pmatrix}$ is the joint covariance matrix.

Equation (2.9) implies that the joint pdf $p\left(\mathbf{x}, y\right)$ is also a mixture of Gaussian. Here the last step in the equation is derived based on equation (4) in the appendix. In Section 2.3 we will see some extensions of the assumption in (2.8), which result in the extended NRBF (ENRBF) model. These extensions will make ENRBF more flexible and efficient. In addition to the parameters that have the correspondences in NRBF, there is an extra parameter σ_j^y appearing in the above ME model. This parameter doesn't directly play a role in the prediction in equation (2.6) because we have taken the expectations in deriving the regression functions. However, σ_j^y could be useful in estimating the confidence interval of a prediction [Wahba, Lin et al. 1999; Hastie, Tibshirani et al. 2001]. All together these parameters $\boldsymbol{\theta} = \left\{\mu_j^y, \boldsymbol{\mu}_j^{\mathbf{x}}, \boldsymbol{\Sigma}_j^{\mathbf{x}}, \sigma_j^y, p(j)\right\}_{j=1}^{M}$ can be estimated by the EM algorithm in a straightforward way.

Based on equation (2.8) and (2.9), there are actually two types of likelihoods that can be used by the EM algorithm in ML estimation, namely the conditional likelihood $\sum_{i=1}^{N} \log p(y_i \mid \mathbf{x}_i)$ and the joint likelihood $\sum_{i=1}^{N} \log p(\mathbf{x}_i, y_i)$. Both of them have been used in the literature, and they are usually dubbed as *discriminative* and *generative* learning respectively. However, in [Xu, Jordan et al. 1995] it has been reported that a significant improvement in convergence rate is achieved by using the joint likelihood with the EM algorithm. So we base our following discussions on the joint likelihood.

In general, the ML estimation corresponds to optimizing the corresponding parameters $\boldsymbol{\theta} = \left\{\mu_j^y, \boldsymbol{\mu}_j^{\mathbf{x}}, \boldsymbol{\Sigma}_j^{\mathbf{x}}, \sigma_j^y, p(j)\right\}_{j=1}^{M}$ by maximizing the joint likelihood $\sum_{i=1}^{N} \log p(\mathbf{x}_i, y_i)$, where $p(\mathbf{x}_i, y_i)$ follows the pdf given in equation (2.9). Instead of optimizing them directly by some nonlinear methods such as the *gradient-based* methods [Bishop 1995], the EM algorithm performs the estimation by alternating between an expectation (E) step and a maximization (M) step. At the E step, the expectation of the likelihood is computed based on some *latent variables* as if they are observed. At the M step, the parameters are re-estimated by maximizing the expected likelihood calculated on the E step. This process is repeated until it converges.

In detail, given the training $data = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ that comprises of n pairs of input-output variables, the ML estimation corresponds to estimating $\boldsymbol{\theta}$ as,

$$\begin{aligned}
\max_{\boldsymbol{\theta}} J &= \max_{\boldsymbol{\theta}} \log p\left(data \mid \boldsymbol{\theta}\right) \\
&= \max_{\boldsymbol{\theta}} \log \prod_{i=1}^{N} p\left(\mathbf{x}_i, y_i \mid \boldsymbol{\theta}\right) \\
&= \max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log p\left(\mathbf{x}_i, y_i \mid \boldsymbol{\theta}\right) \\
&= \max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log\left[\sum_{j=1}^{M} p(j)\, p\left(\mathbf{x}_i, y_i \mid \boldsymbol{\theta}_j, j\right)\right]
\end{aligned} \tag{2.10}$$

where $\boldsymbol{\theta} = \{\mu_j^y, \boldsymbol{\mu}_j^{\mathbf{x}}, \boldsymbol{\Sigma}_j^{\mathbf{x}}, \sigma_j^y, p(j)\}_{j=1}^{M}$. In the above equation the second step is derived according to the *identically and independently distribution* (i.i.d.) assumption of the training data. Here the main difficulty of directly maximizing equation (2.10) arises from the presence of the summation over j in the logarithm. Although nonlinear optimization approaches such as gradient-descent are feasible, the problem can be solved in a neater way by the EM algorithm discussed below.

To start with, we first take a closer look at the parameters inside the logarithm terms, i.e., $\left\{\mu_j^y, \boldsymbol{\mu}_j^{\mathbf{x}}, \boldsymbol{\Sigma}_j^{\mathbf{x}}, \sigma_j^y\right\}_{j=1}^M$. The first-order necessary condition for optimization is $\frac{\partial J}{\partial \boldsymbol{\theta}_j}$=0, where $\frac{\partial J}{\partial \boldsymbol{\theta}_j}$ can be derived as,

$$\begin{aligned}
\frac{\partial J}{\partial \boldsymbol{\theta}_j} &= \frac{\partial}{\partial \boldsymbol{\theta}_j} \sum_{i=1}^N \log\left[\sum_{j=1}^M p(j)\, p\left(\mathbf{x}_i, y_i \,|\, \boldsymbol{\theta}_j, j\right)\right] \\
&= \sum_{i=1}^N \frac{1}{p\left(\mathbf{x}_i, y_i \,|\, \boldsymbol{\theta}\right)} \frac{\partial}{\partial \boldsymbol{\theta}_j} \sum_{j=1}^M p(j)\, p\left(\mathbf{x}_i, y_i \,|\, \boldsymbol{\theta}_j, j\right) \\
&= \sum_{i=1}^N \frac{p(j)}{p\left(\mathbf{x}_i, y_i \,|\, \boldsymbol{\theta}\right)} \frac{\partial}{\partial \boldsymbol{\theta}_j} p\left(\mathbf{x}_i, y_i \,|\, \boldsymbol{\theta}_j, j\right) \\
&= \sum_{i=1}^N \frac{p(j)}{p\left(\mathbf{x}_i, y_i \,|\, \boldsymbol{\theta}\right)} p\left(\mathbf{x}_i, y_i \,|\, \boldsymbol{\theta}_j, j\right) \frac{\partial}{\partial \boldsymbol{\theta}_j} \log p\left(\mathbf{x}_i, y_i \,|\, \boldsymbol{\theta}_j, j\right) \\
&= \sum_{i=1}^N p\left(j \,|\, \mathbf{x}_i, y_i, \boldsymbol{\theta}_j\right) \frac{\partial}{\partial \boldsymbol{\theta}_j} \log p\left(\mathbf{x}_i, y_i \,|\, \boldsymbol{\theta}_j, j\right)
\end{aligned} \tag{2.11}$$

Here the expression of $p\left(\mathbf{x}_i, y_i \,|\, \boldsymbol{\theta}_j, j\right)$ in the last step corresponds to the one given in equation (2.9). Please note that in equation (2.11) the troublesome summation over *j* in (2.10) has been crossed out. Thus, if we know the values of each $p\left(j \,|\, \mathbf{x}_i, y_i, \boldsymbol{\theta}_j\right)$, calculating (2.11) will be much easier since now $p\left(j \,|\, \mathbf{x}_i, y_i, \boldsymbol{\theta}_j\right)$ is a constant and $p\left(\mathbf{x}_i, y_i \,|\, \boldsymbol{\theta}_j, j\right)$ is simply a single Gaussian, of which the derivatives of the parameters are all well known. More importantly, if we take $p\left(j \,|\, \mathbf{x}_i, y_i, \boldsymbol{\theta}_j\right)$ as the latent variables, the EM algorithm enables us to do the computations in (2.11) just by taking the expectations based on $p\left(j \,|\, \mathbf{x}_i, y_i, \boldsymbol{\theta}_j\right)$. In detail, the EM algorithm for NRBF model can be summarized as follows.

Table 2-1: EM algorithm for training NRBF models

1. Initialize the parameters $\boldsymbol{\theta} = \left\{ \mu_j^y, \boldsymbol{\mu}_j^{\mathbf{x}}, \boldsymbol{\Sigma}_j^{\mathbf{x}}, \sigma_j^y, p(j) \right\}_{j=1}^{M}$.

2. Repeat until *convergence*:

(1) E-step: Based on the current value of $\boldsymbol{\theta}$, estimate the expected value of the latent variable π_{ij} as,

$$\pi_{ij} = p\left(j \mid \mathbf{x}_i, y_i, \boldsymbol{\theta}_j \right) = \frac{p\left(\mathbf{x}_i, y_i \mid j, \boldsymbol{\theta}_j\right) p(j)}{\sum_{j'=1}^{M} p\left(\mathbf{x}_i, y_i \mid j', \boldsymbol{\theta}_j\right) p(j')} \tag{2.12}$$

(2) M-step: Based on π_{ij}, re-estimate the parameter $\boldsymbol{\theta}$ by maximizing the likelihood. Based on equation (2.11) and (2.9), the close form representations of the parameters can be derived as,

$$\begin{aligned}
\boldsymbol{\mu}_j^{\mathbf{x}} &= \sum_{i=1}^{N} \pi_{ij} \mathbf{x}_i \Big/ \sum_{i=1}^{N} \pi_{ij} \\
\mu_j^y &= \sum_{i=1}^{N} \pi_{ij} y_i \Big/ \sum_{i=1}^{N} \pi_{ij} \\
\boldsymbol{\Sigma}_j^{\mathbf{x}} &= \sum_{i=1}^{N} \pi_{ij} \left(\mathbf{x}_i - \boldsymbol{\mu}_j^{\mathbf{x}} \right) \left(\mathbf{x}_i - \boldsymbol{\mu}_j^{\mathbf{x}} \right)^T \Big/ \sum_{i=1}^{N} \pi_{ij} \\
\left(\sigma_j^y\right)^2 &= \sum_{i=1}^{N} \pi_{ij} \left(y_i - \mu_j^y \right)^2 \Big/ \sum_{i=1}^{N} \pi_{ij} \\
p(j) &= \frac{1}{n} \sum_{i=1}^{N} \pi_{ij}
\end{aligned} \tag{2.13}$$

After all parameters are determined, equation (2.6) can be used to predict on the new instances. In the above algorithm, the initialization step is not trivial because of the possible presence of *singularities* in the solution [Bishop 2006]. Generally speaking, singularities happen in estimation when the center of one estimated Gaussian components collapses onto a specific training data instance. In that case, the log likelihood we are trying to maximize will go to infinity, which makes the ML estimation become an ill posed problem. To avoid this, the initial parameter values are usually chosen to be random values.

The EM algorithm is guaranteed to converge to a local optimum, which can be proven by *Jensen's inequality* for convex functions [Neal and Hinton 1998]. Compared to a nonlinear optimization method such as gradient-descent, EM's convergence rate is usually faster (with a linear convergence rate). Besides, by maximizing the joint likelihood of both inputs and outputs, the EM trained models usually result in an improved accuracy than the two-step training method. However, just like any nonlinear optimization methods, the EM algorithm could get stuck at a *local* optimum before it converges to the global one.

2.2.3 Main Difficulties of RBF Model

In this section we discuss some main difficulties of RBF model. These difficulties can be viewed as different manifestations of the learning challenges we discussed in Section 1.2; and they directly motivate the extensions of traditional RBF that will be introduced in Section 2.3.

2.2.3.1 Boundary Issue

In Section 2.2.2, we have discussed that the RBF and NRBF models can be viewed as special cases of the more general ME models, and thus can be trained by the EM algorithm. In the EM training stage, with the assistance of latent variables, the training of RBF/NRBF is essentially equivalent to several *local weighted estimations*. To make our viewpoints more clear, recall that at the E step an *instance weight* $\pi_{ij} = p(j \mid \mathbf{x}_i, y_i)$ is estimated for each training instance, which plays the role of the membership function for the *j*th *local neighborhood*. At the M step, a local regression model $f_j(\mathbf{x})$ is fitted by minimizing a *weighted sum-squares* criterion $\sum_{i=1}^{N} \pi_j(\mathbf{x}_i, y_i)\left[y_i - f_j(\mathbf{x}_i)\right]^2$, i.e., the negative log-likelihood. As for an NRBF model, the local fitting $f_j(\mathbf{x})$ is actually assumed to be a constant-value model $f_j(\mathbf{x}) = w_j$ (or μ_j^y), and the instance weights π_{ij} mark the extents of the local neighborhoods. If a Gaussian basis function is used, the shape of the local neighborhood is also Gaussian.

The *boundary issue* is referred to as the problem that a local constant fit (like in NRBF) can be badly biased when a local neighborhood is on the boundaries of the input space [Hastie, Tibshirani et al. 2001]. The bias is mainly caused by the asymmetric distributions of the instances in that neighborhood. As a consequence, the accuracy of the predictions on data points at boundaries is impaired because these predictions are dominated by the local neighborhoods that are closer to boundaries. This kind of prediction bias can also appear in the interior of the space, especially when the instances are distributed unevenly. The boundary issue becomes even more serious when the input dimensionality increases. This is because as a consequence of the curse of dimensionality, the majority of data will be distributed at boundaries in a high dimensional space.

Fortunately, this estimation bias caused by asymmetry can be removed to the first order by replacing a constant $\mathbf{w}$ with a local linear function $\mathbf{w}^T\mathbf{x}$. A linear local fitting coerces a first-order correction to the bias by essentially changing the equivalent kernel. This phenomenon is usually dubbed as *automatic kernel carpentry* in the literature. For more details please refer to [Hastie, Tibshirani et al. 2001]. In Section 2.3 we will see that an effective extension of NRBF proposed by [Xu 1998] is essentially based on this idea of exploiting a linear model in local estimations.

Although it is possible to further reduce the estimation bias by fitting a higher order local model (e.g., a quadratic), the linear fitting is usually preferred in practice. This is because unlike a linear fitting that reduces estimation bias dramatically on boundaries at a modest cost in estimation variance, a higher order local fitting does little at boundaries for bias, but usually increases the variance significantly [Hastie, Tibshirani et al. 2001]. As a consequence, a high estimation variance usually results in the degeneration of a model's generalization ability (i.e., the prediction ability on new data).

2.2.3.2 Explosion of Basis Function

Another problem of NRBF/RBF models is that the number of their basis functions could increase exponentially with input dimensionality, because in this case the high dimensional data tend to be distributed more sparsely and thus more basis functions are needed to cover the training data. This is actually another aspect of the curse of dimensionality. The *explosion of basis functions* brings up a huge computation cost and could prevent NRBF/RBF models from being used in practice. Even worse, using more basis functions inevitably increases the model complexity whereas the data complexity remains the same. As a consequence, the over-complicated model is easy to over-fit and tends to give poor prediction performance.

In the next section, we will see that this difficulty can also be alleviated by using a higher order fitting like in the case of boundary issues. By bringing extra flexibilities, a linear local fitting is able to model a complicated input-output relationship in a more adaptive way. Thus, it usually uses much fewer basis functions and can provide a comparable or even better accuracy. Although high order fittings introduce extra parameters, the computation cost is usually reduced in practice. This is because the reduced number of the basis functions usually compensates for the cost of more parameters. We will discuss this in details in the following section.

2.3 ENRBF and EM Learning

As introduced above, the main difficulties for traditional RBF/NRBF models are the *boundary issue* and the *explosion of basis function* in high dimensional spaces. The consequences usually include an exponential increase of the computation cost and an impairment of learning accuracy. In the literature, several modifications of RBF/NRBF have been proposed to overcome these weaknesses with different focuses [Bugmann 1998; Xu 1998; Albrecht, Busch et al. 2000; Looney 2002; Marcelino, Ignacio et al. 2003; Staiano, Tagliaferri et al. 2006]. Particularly, Xu [Xu 1998; Xu 2004] proposed the *Extended Normalized Radial Basis Function* (ENRBF) model and an associated EM-based training algorithm. Compared to the traditional models, ENRBF has an improvement in learning accuracy and a reduction in number of basis functions. These advantages are significant especially when the input dimensionality of data is high. In this section, we will introduce the basic assumptions underlying ENRBF and its EM based training algorithm. Its advantages over the traditional models will be explored in the next section.

From a general viewpoint, both NRBF and ENRBF can be viewed as the special cases of the mixture-of-expert (ME) model [Jordan and Jacobs 1994] discussed in Section 2.2.2. Recall that an ME model is based on the following assumption of the input-output relations,

$$p\left(y \mid \mathbf{x}\right) = \sum_{j=1}^{M} p\left(y \mid \mathbf{x}, j\right) p\left(j \mid \mathbf{x}\right) \tag{2.14}$$

Here, the conditional probability $p(y\,|\,\mathbf{x})$ is represented by a mixture of M components. In a regression problem, this probability can be used to calculate a point estimate of outputs, i.e.,

$$f(\mathbf{x}) = E[y\,|\,\mathbf{x}] = \sum_{j=1}^{M} \mu_j^y p(j\,|\,\mathbf{x}) \tag{2.15}$$

The details of this derivation can be found in Section 2.2.2. Here $f(\mathbf{x})$ is called regression function and is actually the mean function of outputs under the distribution $p(y\,|\,\mathbf{x})$. In the above equation, $\mu_j^y = E[y\,|\,\mathbf{x}, j]$ is the conditional mean of the outputs in the *j*th component, and $p(j\,|\,\mathbf{x})$ can is the posterior probability that the *j*th component generates an instance $\mathbf{x}$.

Under this framework, the NRBF model can be viewed as a special case of ME by assuming $\mu_j^y = w_j$ and $p(j\,|\,\mathbf{x}) = \tilde{\phi}(\mathbf{x}\,|\,\boldsymbol{\theta}_j)$, which results in,

$$f(\mathbf{x}) = \sum_{j=1}^{M} w_j \tilde{\phi}(\mathbf{x}\,|\,\boldsymbol{\theta}_j) \tag{2.16}$$

Here $\tilde{\phi}(\mathbf{x}\,|\,\boldsymbol{\theta}_j)$ is the normalized basis function of NRBF and w_j is the corresponding coefficient. w_j is a constant parameter that is independent of inputs.

The extended NRBF (ENRBF) [Xu 1998] resembles NRBF in the way that it can also be viewed as a special case of ME, with the same assumption of $p(j\,|\,\mathbf{x})$ but a different assumption of μ_j^y. In ENRBF, $\mu_j^y = \mathbf{w}_j^T \mathbf{z}$, where $\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$ is the *augmented input vector*. Thus, an ENRBF model is represented by,

$$f(\mathbf{x}) = \sum_{j=1}^{M} (\mathbf{w}_j^T \mathbf{z}) \tilde{\phi}(\mathbf{x}\,|\,\boldsymbol{\theta}_j) \tag{2.17}$$

Here this linear form used in each local basis function is similar to the first-order Takagi-Sugeno-Kang (TSK) fuzzy model [Takagi and Sugeno 1985; Sugeno and Kang 1988]. In fact, ENRBF and the first-order TSK has a very similar structure. For a comparison of the two please refer to [Ma, Abdul et al. 2006 A; Ma, Abdul et al. 2006 B].

The connection between ENRBF and ME suggests an EM-based training algorithm similar to the one used to train NRBF in Section 2.2.2. To illustrate the algorithm detail, we further make a specific assumption for the probabilities $p(\mathbf{x} \mid j)$ and $p(y \mid \mathbf{x}, j)$. For example, assume a Gaussian distribution,

$$p(\mathbf{x} \mid j) = \mathrm{N}\left(\mathbf{x}; \boldsymbol{\mu}_j^x, \boldsymbol{\Sigma}_j^x\right) \tag{2.18}$$

This assumption also corresponds to assuming Gaussian shaped basis functions, because of the relations between $p(\mathbf{x} \mid j)$ and the normalized basis functions $\tilde{\phi}(\mathbf{x} \mid \boldsymbol{\theta}_j) = p(j \mid \mathbf{x})$ $= p(\mathbf{x} \mid j)\, p(j) \Big/ \sum_{j'=1}^{M} p(\mathbf{x} \mid j')\, p(j')$. Furthermore, as discussed in Section 1.3, for regression cases we can make a further assumption of $p(y \mid \mathbf{x}, j)$,

$$\begin{aligned} p(y \mid \mathbf{x}, j) &= \mathrm{N}\left(y; \mu_j^y, \left(\sigma_j^y\right)^2\right) \\ &= \mathrm{N}\left(y; \mathbf{w}_j^T \mathbf{z}, \left(\sigma_j^y\right)^2\right) \end{aligned} \tag{2.19}$$

here we have used the local linear form assumption in ENRBF that $\mu_j^y = \mathbf{w}_j^T \mathbf{z}$. Equation (2.19) is rational for regression cases because the outputs in regression problems are real values. On the other hand, in Section 3.1 we will see that this assumption is not appropriate assumption anymore when we come to classifications.

To summarize, equations (2.17)~(2.19) form the underlying assumptions for the ENRBF model. To estimate the parameters $\boldsymbol{\theta} = \left\{ \boldsymbol{\mu}_j^{\mathbf{x}}, \boldsymbol{\Sigma}_j^{\mathbf{x}}, \mathbf{w}_j^T, \sigma_j^y, p(j) \right\}_{j=1}^{M}$ in these equations, the EM algorithm can be applied by maximizing the joint likelihood,

$$\begin{aligned} p(\mathbf{X}, \mathbf{y} \mid \boldsymbol{\theta}) &= \prod_{i=1}^{N} p(\mathbf{x}_i, y_i \mid \boldsymbol{\theta}) \\ &= \prod_{i=1}^{N} \sum_{j=1}^{M} p(\mathbf{x}_i, y_i \mid j, \boldsymbol{\theta})\, p(j \mid \boldsymbol{\theta}) \\ &= \prod_{i=1}^{N} \sum_{j=1}^{M} p(y_i \mid \mathbf{x}_i, j, \boldsymbol{\theta})\, p(\mathbf{x}_i \mid j, \boldsymbol{\theta})\, p(j \mid \boldsymbol{\theta}) \end{aligned} \tag{2.20}$$

Here the key in applying the EM algorithm is to estimate the posterior probability $p(j \mid \mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta})$, which plays the role of latent variables. With these latent variables, the estimation of a mixture of Gaussians can be converted to an iterative process of several separate estimations of single Gaussians. Please refer to Table 2-1 for more detail about the EM algorithm.

When training ENRBF in practice, the EM algorithm is usually preferred although alternatives exist such as the gradient based optimization [Fletcher 1987; Nocedal and Wright 1999] or the similiar two-step training algorithms originating from NRBF. Compared to these alternatives, the EM algorithm has a comparable accuracy with the gradient based nonlinear optimization and at the same time remains as fast as the two-step algorithm. Moreover, since the EM algorithm has a clear statistical interpretation at each step, it is more explainable to the human experts. These benefits make the EM algorithm more favorable than the other black-box methods, especially when explanations of the models are as important as accurate results.

2.4 Advantages of ENRBF in High Dimensional Space

In this section, we take a closer look at how ENRBF model can help alleviate the difficulties with the traditional NRBF/RBF models, namely the boundary issue and the explosion of basis functions. As discussed in the last two sections, the EM algorithms for NRBF and ENRBF can both be viewed as fitting local estimations in the local neighborhoods specified by the basis functions. The main difference between NRBF and ENRBF lies in the different assumptions of their local fittings, i.e., the form of $f_j(\mathbf{x})$ for each neighborhood. In NRBF, the local fitting $f_j(\mathbf{x})$ is assumed as a constant function w_j (in equation (2.16)) that is independent of input $\mathbf{x}$. On the other hand, $f_j(\mathbf{x})$ in ENRBF is represented by a linear model $\mathbf{w}_j^T\mathbf{z}$ of inputs (in equation (2.17)).

Based on our discussion in Section 2.2.3.1, a linear local fitting can help alleviate the boundary issue through the *automatic kernel carpentry*. Furthermore, a linear fitting could even achieve a better learning accuracy than those higher order polynomial, as the linear fitting essentially achieves a good balance between the estimation bias and variance [Hastie, Tibshirani et al. 2001].

The linear local assumption in the ENRBF model also relieves the explosion of basis functions in high dimensional learning tasks. Compared to the constant fitting used by NRBF, a linear local fitting is able to model the underlying input-output relation in a more adaptive way. For example, for a regression task where the underlying input-output function is actually linear, principally ENRBF is able to fit the data with one basis function. On the other hand, NRBF may need several overlapped and contiguous basis functions to achieve the same accuracy. Increasing the order of the local fitting could further reduce the number of basis functions, but there is a risk of introducing too many parameters that finally leads to an over-fitting. Besides, the EM algorithm for higher order fittings will be much more complicated than the one used to estimate the linear form in ENRBF, which forfeits its nature of easy implementation and fast execution. Although a local linear model also introduces extra parameters, usually the training time of ENRBF will not significantly increase. This is because the EM steps for a linear estimation still remains simple and fast. In fact, by using fewer and more flexible local models, the training time of ENRBF is usually reduced in practice, especially in high dimensional applications. The advantages of ENRBF will be further demonstrated in the experiments in Section 2.5 and Chapter 3.

2.5 An Experimental Study

In this section three regression experiments are constructed to show the advantages of ENRBF and its EM training algorithm. We will compare ENRBF trained by EM with the traditional NRBF trained by the two-step algorithm. Gaussian shaped basis functions are used in both models. These basis functions are parameterized by their centers and diagonal covariance matrices.

As some experiments are directly from real-world applications, different inputs typically have very different value ranges. To avoid undesired dominations of a specific input feature and the consequent numerical difficulties, all inputs will be *normalized* to zero mean and unit variance before being presented to the learning models. To avoid over-fitting, we take the traditional model selection method, namely the *three-fold cross validation*, to estimate the optimal model complexity. In this case the model complexity of NRBF and ENRBF are represented by the number of components (basis functions) used in the models. In detail, we fit a group of the same type models with different complexities; the optimal model will be selected as the one that has the smallest complexity, and at the same time satisfies the accuracy condition that its cross-validation accuracy is within one standard error of the best accuracy in the group [Hastie, Tibshirani et al. 2001]. The *Root Mean Squared Error* (RMSE) is used as the accuracy measures for regression problems.

2.5.1 Artificial Regression

The first experiment is based on a simple regression problem. It main purpose is to demonstrate how ENRBF and NRBF work. The input dimensionality is exactly 1, thus it is convenient to plot the results in a 2-D figure. The dataset consists of 200 training instances generated by the univariate nonlinear function $y = \sin^2(2\pi x) + \varepsilon$ where ε is the additive noise following the zero mean Gaussian and the input values x are generated by following the *uniform distribution* in the $[0,1]$ interval. The signal-to-noise ratio (SNR) level is specified as 2.

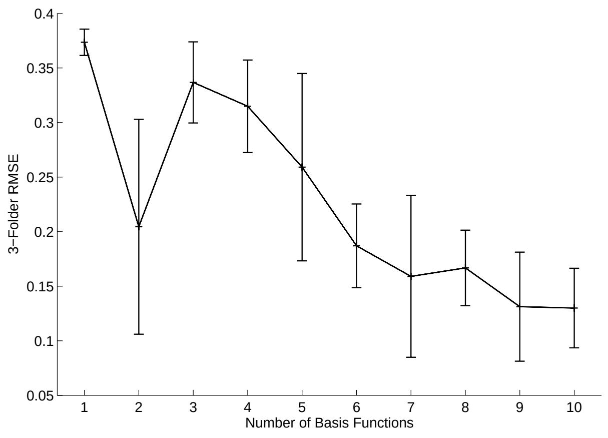


Figure 2-1:Cross validation errors of the two-stage NRBF, the optimal number of basis functions is 7.

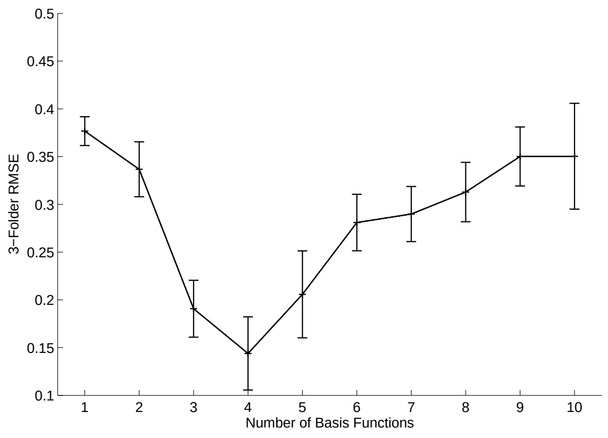


Figure 2-2: Cross validation errors of ENRBF, the optimal number of basis functions is 4.

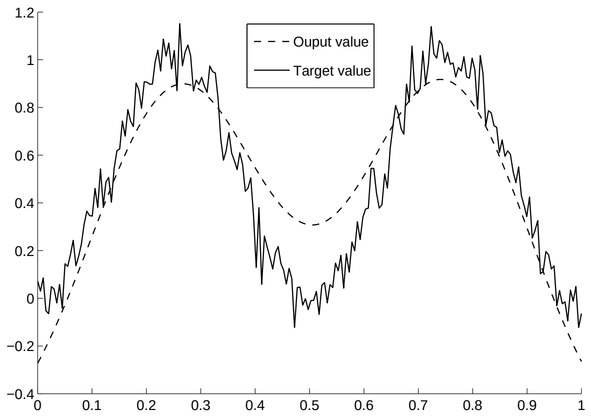


Figure 2-3: Outputs of two-stage NRBF v.s. targets

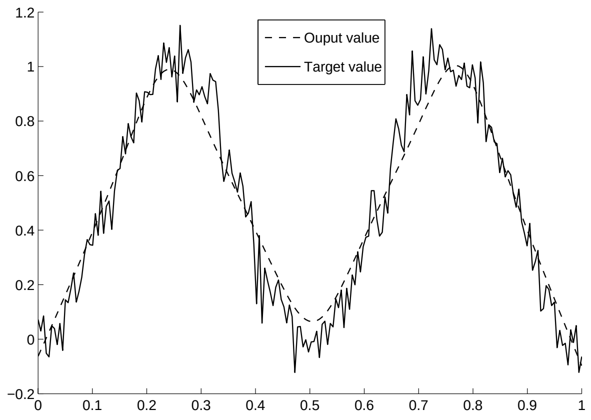


Figure 2-4: Outputs of ENRBF v.s. targets

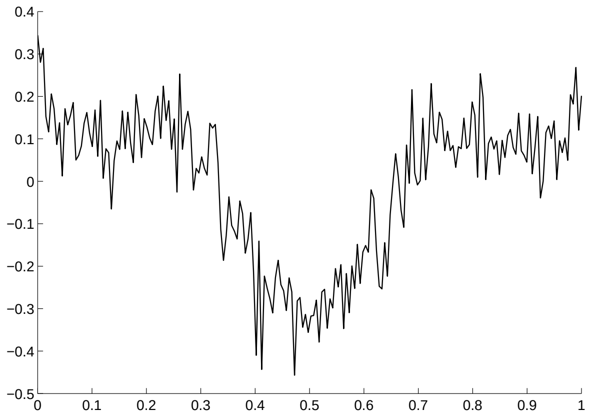


Figure 2-5: Differences of Targets and Predicted Outputs of two-stage NRBF

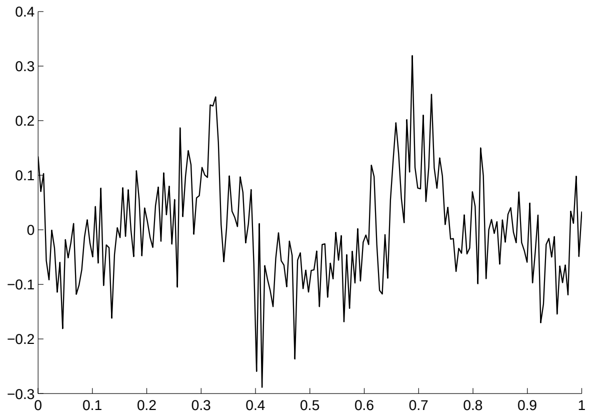


Figure 2-6: Differences of Targets and Predicted Outputs of ENRBF

The comparisons of the results from both NRBF and ENRBF are shown in Figure 2-1~Figure 2-6. Figure 2-1 and Figure 2-2 show the model selection results based on the three-fold cross validation. To do the model selection, ten models of each type with different number of basis functions (from 1 to 10) are trained. The optimal model of each type is selected as the one with the smallest number of basis functions and has the error within one standard interval of the lowest. For NRBF and ENRBF, the optimal model complexities are 7 and 4 respectively. We can see that even for this simple problem, the number of basis functions used in NRBF is almost twice of the one in ENRBF.

After determining the appropriate complexity for each type, the performances of the optimal NRBF model and the ENBRF model are further compared in Figure 2-3 and Figure 2-4. Here the RMSEs of both models are close to the noise deviation as expected, which indicates the over-fitting doesn't happen. However, there are also differences between the results from the two models. For example, although both models have observable biases at the curvature part of the target function (a phenomenon observed as *trimming the hills* and *filling the valleys* in the literature [Hastie, Tibshirani et al. 2001]), the bias of ENRBF is much smaller than NRBF at the boundaries (near 0 or 1). This observation is shown more clearly in Figure 2-5 and Figure 2-6, where the differences between the targets and the model predictions are plotted. The reason why ENRBF has a smaller bias is because it can handle the *boundary issue* better as we have discussed in Section 2.4.

Because the input dimensionality is only one, ENRBF's impact on the explosion of basis functions is not observed. We will illustrate this in the following experiments.

2.5.2 ICU Ventilation Control

To further explore the advantages of ENRBF, a medical regression problem from the real-world application is used in this section to compare the two models. The task is to predict the time series of the *Fraction of Inspired Oxygen* (FiO2) values used for ventilation control. In Intensive Care Unit (ICU), the ventilators play an important role in the treatment of patients. There have been several computer-based models proposed in the literature for the purpose of the automatic ventilators control [Wang, Shaw et al. 1998; Kwok, Linkens et al. 2003]. In this experiment, to predict the one-hour-later value of FiO2, four variables are used as the inputs, namely RR (Respiratory Rate), SaO2 (Oxygen Saturation), PEEP (Positive End Expiratory Pressure) and the current FiO2 (Fraction of Inspired Oxygen). In the training set, there are 408 observations collected from a 20-day record of an individual patient under the Biphasic Positive Airway Pressure (BIPAP) ventilation mode, from the Kandang Kerban Women's and Children's Hospital in Singapore.

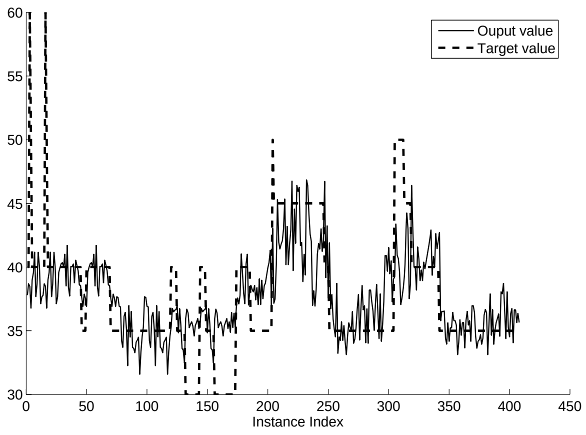


Figure 2-7: Predictions of two-stage NRBF

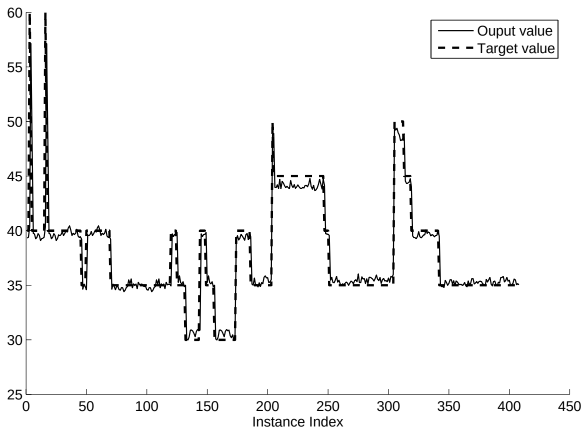


Figure 2-8: Predictions of ENRBF

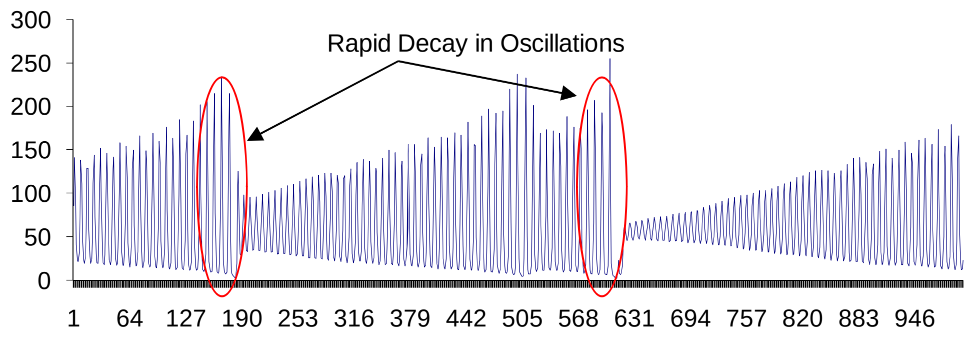
Table 2-2: Comparison of ENRBF and NRBF on ventilation data

	# of Basis Functions	*RMSE*	*Training Time (secs)*
Two-stage NRBF	6	3.68	0.13
ENRBF	1	1.94	0.02

Figure 2-7 and Figure 2-8 show the predictions from NRBF and ENRBF respectively, compared to the target time series. Both results are from the respective optimal models that are selected by the three-fold cross validation. From Figure 2-7, we can see although the trend of the series is roughly captured by NRBF, a lot of details have been lost. Increasing the number of basis functions can increase the modeling accuracy for the training data, but will not help in general due to the degeneration of overall accuracy caused by over-fitting. On the other hand, there is a significant improvement in the approximation by ENRBF, which is plotted in Figure 2-8. This observation can also be verified by the comparison of RMSE from the two models that are present in Table 2-2. Although NRBF uses 6 parameters for each basis function (5 basis function parameters plus 1 constant coefficient) and ENRBF uses 10, NRBF actually use more parameters than ENRBF, as the number of basis functions in NRBF is six times of the one used in ENRBF. Thus, ENRBF gives better accuracy not because it uses more parameters. Its improvement of accuracy is mainly due to its extra flexibility introduced by its linear model assumption, as we have discussed in the above sections. Furthermore, ENRBF even has a shorter training time than NRBF trained by the fast two-step algorithm. This demonstrates ENRBF's potentials in reducing the computation costs.

2.5.3 Laser Time-Series Prediction

To illustrate how ENRBF alleviates the explosion of basis functions, in this last regression experiment we use a regression problem that has a variational input dimensionality.



The dataset consists of 1000 observations of the fluctuations in a far-infrared (FIR) laser (which is online available at http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe.html). The task is to do the *one-step-ahead prediction* using the past Δt observation values, i.e., to predict the observation at time $t+1$ by the observations at t, $t-1 \ldots t-\Delta t$. Here the challenge is to appropriately model the rapid turnovers in the time series.

When a specific value of Δt is given, the original prediction problem corresponds to a regression problem, where the inputs are $x_{t-\Delta t}, x_{t-\Delta t+1}, \ldots, x_t$ and the target output is x_{t+1}. The history length Δt just corresponds to the input dimensionality of the regression problem. Thus, by using different Δt values in the experiment, we can construct a series of regression problems with a similar structure and different input dimensionalities. This provides us a good opportunity to observe how the different models handle the increase of the input dimensionality.

Table 2-3: Comparison of ENRBF and NRBF on laser data

Input Dim. Δt	*Gradient-based NRBF*					*ENRBF*			
	RMSE	*Optimal Complexity*	*Increasing Ratio of Complexity*	*Training Time (secs.)*		*RMSE*	*Optimal Complexity*	*Increasing Ratio of Complexity*	*Training Time (secs.)*
1	38.37	9	-	36.23		37.88	5	-	81.33
2	8.83	14	$14/9 \approx 1.6$	50.77		8.50	9	$9/5 \approx 1.8$	111.32
4	7.87	28	$28/14 \approx 2$	103.39		7.62	15	$15/9 \approx 1.7$	193.14
8	10.13	32	$32/28 \approx 1.1$	234.16		10.17	19	$19/15 \approx 1.3$	264.22
16	15.90	54	$54/32 \approx 1.7$	733.48		16.03	20	$20/19 \approx 1.0$	323.86
32	20.49	80	$80/54 \approx 1.5$	881.84		20.70	24	$24/20 \approx 1.2$	365.99

The result comparisons under different Δt values are summarized in Table 2-3. In the table the model complexity is measured in terms of the number of basis functions, and is optimally selected by the three-fold cross validation. As we mentioned in the beginning of this experiment, the rapid oscillations make the approximation of the time series a complicated problem. Thus, the training times of the models seem significantly longer than those in the other experiments. Moreover, because here the prediction errors of NRBF trained by the two-step algorithm are unacceptably high, we turn to using the more complicated gradient-based training algorithm to seek better accuracies. As a supervised nonlinear optimization method, the gradient-based training algorithm can significantly increase NRBF's accuracy, but at a cost of a longer training time.

From Table 2-3 we can see that at each comparable error level, ENRBF usually uses much fewer basis functions than NRBF. This phenomenon becomes more obvious along with the increase of the input dimensionality Δt. As a direct result, the training time of ENRBF is much reduced than NRBF. We are also interested in the *increasing rates* of the model complexity (in terms of the number of basis functions) with the increase of the input dimensionality. A constantly high increasing rate with input dimensionality usually indicates the explosion of the basis functions, as the result of suffering the curse of dimensionality. From the results, it can be observed that the increasing rate of NRBF keeps at the same level even after the RMSE gets very high, although at that time the dimensionality has become too high such that the number of training data is not enough to approximate the underlying high dimensional function anymore. On the other hand, ENRBF is able to slow down the increasing rate of complexity to reflect this situation of the learning saturation. In this way the *explosion of basis functions* discussed in Section 2.4 is effectively alleviated by ENRBF.

From the results we can also observe that it looks as if there is an optimal value for Δt where the RMSEs for both models are lowest. Based on the result table we can tell this optimal value is about 4. Surrounding this optimal value, the models' RMSEs decrease first and then increase again. An intuitive explanation is as follows. Before Δt arriving at this optimal value, the models' prediction errors first decrease along with the increase of Δt, since a longer history and thus extra information have been involved for prediction. Beyond this optimal value, although there are even more information involved, the error rates start to increase as a result that the training data sample begins to become too scarce for modeling such a complicated underlying function in a high dimensional space. This essentially provides us a vivid picture of how the *curse of dimensionality* affects the learning models.

2.6 Summary

In this chapter, we reviewed the two main learning paradigms in the literature. Following this general discussion, we further introduced a specific parametric model—the RBF model. Compared to other nonlinear parametric models, RBF is easy to implement and has a relatively fast training algorithm. This advantage makes it one of the best off-the-shelf learning models in practice.

Table 2-4: List of popular extensions of RBF

Original Model	*Extended Models*	*Classification Model?*	*Reduced Computation Cost?*
RBF (parametric)	Self-Organized RBF [Albrecht, Busch et al. 2000]	√	
	Gaussian Mixture Classifier (Chapter 3)	√	
	Common-Basis GMC (Chapter 3)	√	
	Two-stage NRBF (Chapter 3)	√	
	Bayesian Ying-Yang RBF [Xu 2004]		√
	Extended Normalized RBF [Xu 1998]		√
	ENRBF Classifier (Chapter 3)	√	√

However, as discussed in Section 2.2.3, RBF has some limitations that prevent it from being directly used in high-dimensional learning tasks. These limitations include the performance degeneration and the basis function explosion caused by the increasing number of input features. These are essentially different reflections of the curse of dimensionality introduced in Chapter 1. In the literature, different extensions of RBF models have been proposed to overcome these limitations. Some of them are summarized in Table 2-4. Among these researches, a lot of efforts have gone to the attempts to reduce the computation costs involved in training and evaluations stages. These modified models usually have a reduced computation cost compared to the traditional models, especially in high-dimensional learning applications. Thus, these models are usually called *computationally efficient learning models*. Among these models, we specially introduce the ENRBF model proposed by [Xu 1998], as it forms the starting point of our own research that will be introduced in the next chapter. Since Xu's original paper lacks a systemic comparison of the traditional RBF and ENRBF, both a theoretical discussion and some experimental analysis are provided in this chapter.

Although these computationally efficient models usually have a significant improvement for regression problems, the extensions of them for classification problems are not straightforward. As introduced in Chapter 1, regression and classification problems have very different assumptions and thus learning models for one type cannot be directly used for the other. In order to further handle the challenges in high dimensional classification problems, we propose a new extension—the ENRBF Classifier (ENRBFC) in the next chapter. It can be viewed as a further enhancement of the ENRBF model for classification problems.

Chapter 3

3 Modification of the ENRBF Regression Model for Classification Problems

In the last chapter we reviewed the main difficulties of the traditional RBF model when it handles high dimensional learning problems. We also introduced how the ENRBF model [Xu 1998], as an extension of RBF for regression problems, helps alleviate these difficulties. In this chapter, we take a further step to propose a modification of the ENRBF model, namely the ENRBF Classifier (ENRBFC) model, to handle the high dimensional classification problems. Furthermore, an EM based training algorithm is proposed to train the ENRBFC model. Some experiments are constructed to compare the performance of ENRBFC with other peer classifiers, which show the potentials of the proposed methods.

3.1 ENRBF Classifier

In this section we present the basic assumptions under the ENRBFC model. Similar to NRBF and ENRBF, the proposed *ENRBF Classifier* (ENRBFC) also bears a close connection to the general ME model introduced in Section 2.3. Inspired by this connection, we also propose a modified EM algorithm to train ENRBFC. In Section 3.2 we will also review several similar classifiers that will be compared with ENRBFC.

3.1.1 ENRBFC Architecture

As discussed in Section 1.3, the main obstacle that prevents regression models from being used directly for classifications is the different output types of regressions and classifications. As a result, the output assumptions for ENRBF in equation (2.19) is not appropriate anymore when we come to classification problems, because generally the *1-of-k* encoded classification outputs are discrete values and cannot be modeled by a Gaussian distribution that generates real values. In this case, a more appropriate assumption for the *j*th expert $p(y \mid \mathbf{x}, j)$ of an ME model is a *logistic*/*softmax regression* model [Bishop 2006]. In the following discussions, we will take a binary classification as an example, and these discussions can be easily extended to multi-classification cases.

In a binary classification problem, the outputs are usually encoded as $\{-1,1\}$. This is different from real values assumption used in regressions. To reflect this difference, we use a different assumption, namely the *logistic regression* model, for the *j*th expert of the ME framework, i.e.,

$$\begin{aligned} p(y=1 \mid \mathbf{x}, j) &= \sigma\left(\mathbf{w}_j^T \mathbf{z}\right) \\ p(y=-1 \mid \mathbf{x}, j) &= 1-\sigma\left(\mathbf{w}_j^T \mathbf{z}\right) \end{aligned} \tag{3.1}$$

where $\sigma(a) = \dfrac{1}{1+\exp(-a)}$ is the *logistic* function and $\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$ is the augmented input vector. As a squashing function, the logistic function maps an arbitrary real value to the $[0,1]$ interval. As for a multi-classification problem, a more appropriate assumption of $p(y \mid \mathbf{x}, j)$ will be the *softmax* model $\exp\left(\mathbf{w}_{kj}^T \mathbf{z}\right) \Big/ \sum_{k'=1}^{C} \exp\left(\mathbf{w}_{k'j}^T \mathbf{z}\right)$, in which C is the total number of classes.

Although the *expert models* in ENRBFC and ENRBF are different, their *gating functions* are the same, i.e.,

$$\begin{aligned} p(j \mid \mathbf{x}) &= \tilde{\phi}\left(\mathbf{x} \mid \boldsymbol{\theta}_j\right) \\ &= \frac{p(\mathbf{x} \mid j)\, p(j)}{\sum_{j'=1}^{M} p(\mathbf{x} \mid j')\, p(j')} \end{aligned} \tag{3.2}$$

where the input *probability density distribution* (pdf) $p(\mathbf{x} \mid j)$ can be assumed to follow a Gaussian distribution,

$$p(\mathbf{x} \mid j) = \mathrm{N}\left(\mathbf{x}; \boldsymbol{\mu}_j^{\mathbf{x}}, \boldsymbol{\Sigma}_j^{\mathbf{x}}\right) \tag{3.3}$$

Thus, based on the ME framework, the ENRBFC model can be represented as

$$\begin{aligned} p(y=1 \mid \mathbf{x}) &= \sum_{j=1}^{M} p(y=1 \mid j, \mathbf{x})\, p(j \mid \mathbf{x}) \\ &= \sum_{j=1}^{M} \sigma\left(\mathbf{w}_j^T \mathbf{z}\right) \tilde{\phi}\left(\mathbf{x} \mid \boldsymbol{\theta}_j\right) \end{aligned} \tag{3.4}$$

This corresponds to a special case of ME with a logistic regression expert model $\sigma\left(\mathbf{w}_j^T \mathbf{z}\right)$ and a specific gating function $\tilde{\phi}\left(\mathbf{x} \mid \boldsymbol{\theta}_j\right)$. Compared to the ENRBF model represented in equation (2.17), here their main difference of the two is that the real-valued output $E[y \mid \mathbf{x}, j]$ has been replaced by the logistic regression output $p(y=1 \mid j, \mathbf{x})$. Thus, in a classification problem, instead of predicting the output values $E[y \mid \mathbf{x}]$ directly, the classifiers usually predict the conditional probabilities $p(y \mid \mathbf{x})$ for different output values. This trick will also be in Chapter 4 when we develop the Gaussian Process Classification models.

3.1.2 EM-based ENRBFC Training Algorithm

In this section we present an EM based training algorithm for the ENRBFC model, which is based on its connection to the ME model. Without loss of generality, we will stick to the discussion of a binary classification problem, where the output y is encoded as $\{-1, 1\}$. This encoding scheme brings a pleasant result that $p(y \mid \mathbf{x}, j)$ in the logistic function can be simply calculated as $\sigma\left(y\mathbf{w}_j^T\mathbf{z}\right)$, which is based on a property of the logistic function that $\sigma(-a)=1-\sigma(a)$.

In equation (3.4) the parameters for an ENRBFC model are represented as $\boldsymbol{\theta}=\left\{p(j),\boldsymbol{\mu}_j^{\mathbf{x}},\boldsymbol{\Sigma}_j^{\mathbf{x}},\mathbf{w}_j\right\}_{j=1}^{M}$. In an EM based training algorithm, these parameters are optimized by the maximization of the joint likelihood $\sum_{i=1}^{N} p(y_i\,|\,\mathbf{x}_i)\,p(\mathbf{x}_i)$. At the *expectation* step, the weight of the *i*th instance with respect to the *j*th component, i.e., the *latent variable* $p(j\,|\,\mathbf{x}_i,y_i)$ will be estimated based on the current values of the parameters. At the *maximization* step, with the assistance of these latent variables, the maximization of the joint likelihood will be performed by doing several separate maximum likelihood estimations of single Gaussians. These two steps will be iterated until the updates of the parameters converge.

In details, the EM algorithm for ENRBFC is summarized in Table 3-1.

Table 3-1: EM training algorithm for ENRBFC model

1. Initialize the parameters $\boldsymbol{\theta}=\left\{p(j),\boldsymbol{\mu}_j^{\mathbf{x}},\boldsymbol{\Sigma}_j^{\mathbf{x}},\mathbf{w}_j\right\}_{j=1}^{M}$.
2. Repeat until *convergence*:   (1) E-step: Based on the current value of $\boldsymbol{\theta}$, estimate the expected value of the latent variable π_{ij} as,

$$
\begin{aligned}
\pi_{ij} &= p(j \mid \mathbf{x}_i, y_i) \\
&= \frac{p(j)\,p(\mathbf{x}_i \mid j)\,p(y_i \mid \mathbf{x}_i, j)}{\sum_{j'=1}^{M} p(j')\,p(\mathbf{x}_i \mid j')\,p(y_i \mid \mathbf{x}_i, j')} \\
&= \frac{p(j)\,p(\mathbf{x}_i \mid j)\,\sigma(y_i \mathbf{w}_j^T \mathbf{z}_i)}{\sum_{j'=1}^{M} p(j')\,p(\mathbf{x}_i \mid j')\,\sigma(y_i \mathbf{w}_{j'}^T \mathbf{z}_i)}
\end{aligned}
\tag{3.5}
$$

(2) M-step: Based on π_{ij}, re-estimate the parameter $\boldsymbol{\theta}$ by maximizing the log-likelihood, that is,

$$
\boldsymbol{\theta}^{new} = \arg\min_{\boldsymbol{\theta}} \sum_{j=1}^{M} \sum_{i=1}^{N} p(j \mid \mathbf{x}_i, y_i, \boldsymbol{\theta}) \log\left[p(j \mid \boldsymbol{\theta})\, p(\mathbf{x}_i \mid j, \boldsymbol{\theta})\, p(y_i \mid \mathbf{x}_i, j, \boldsymbol{\theta}) \right] \tag{3.6}
$$

With the assistance of $p(j \mid \mathbf{x}_i, y_i)$ estimated at the E-step, the optimal values of $p(j)$, $\boldsymbol{\mu}_j^{\mathbf{x}}$ and $\boldsymbol{\Sigma}_j^{\mathbf{x}}$ can be derived in close-form representations,

$$
p(j)\mid_{new} = \frac{1}{N} \sum_{i=1}^{N} p(j \mid \mathbf{x}_i, y_i) \tag{3.7}
$$

$$
\boldsymbol{\mu}_j^{\mathbf{x}} \mid_{new} = \sum_{i=1}^{N} p(j \mid \mathbf{x}_i, y_i)\, \mathbf{x}_i \Big/ \sum_{i=1}^{N} p(j \mid \mathbf{x}_i, y_i) \tag{3.8}
$$

$$
\boldsymbol{\Sigma}_j^{\mathbf{x}} \mid_{new} = \sum_{i=1}^{N} p(j \mid \mathbf{x}_i, y_i) (\mathbf{x}_i - \boldsymbol{\mu}_j^{\mathbf{x}})(\mathbf{x}_i - \boldsymbol{\mu}_j^{\mathbf{x}})^T \Big/ \sum_{i=1}^{N} p(j \mid \mathbf{x}_i, y_i) \tag{3.9}
$$

However, the close-form representation of $\mathbf{w}_j \mid_{new}$ cannot be derived in general due to the nonlinearity of the logistic function. Thus, we use the fast *Iterative Reweighted Least Squares* (IRLS) algorithm [Rubin 1983; Bishop 2006] to iteratively estimate its optimal values. The IRLS method optimizes the parameters based on the *Newton-Raphson* algorithm [Fletcher 1987], in details,

$$\mathbf{w}_j|_{new} = \mathbf{w}_j|_{old} - \mathbf{H}^{-1}\mathbf{g} \tag{3.10}$$

where

$$\mathbf{H} = \mathbf{X}^T \boldsymbol{\Lambda}_j \mathbf{R} \mathbf{X} \tag{3.11}$$

$$\mathbf{g} = \mathbf{X}^T \boldsymbol{\Lambda}_j \left(\hat{\mathbf{y}} - \frac{(\mathbf{y}+1)}{2} \right) \tag{3.12}$$

Here $\mathbf{H}$ is the *Hessian* matrix and $\mathbf{g}$ is the *gradient*. In the above equations, $\mathbf{X}$ is the design matrix, $\boldsymbol{\Lambda}_j$ is the diagonal matrix composed of the instance weights $\boldsymbol{\Lambda}_j = \text{diag}\left(\left[p(j\,|\,\mathbf{x}_1, y_1), \ \ldots \ , p(j\,|\,\mathbf{x}_N, y_N)\right]\right)$. $\mathbf{R}$ is also a diagonal matrix with $\mathbf{R} = \text{diag}\left(\left[\hat{y}_1(1-\hat{y}_1), \ \ldots \ , \hat{y}_N(1-\hat{y}_N)\right]\right)$, in which $\hat{y}_i$ is the model's prediction value for training input $\mathbf{x}_i$. $\hat{y}_i$ is always in the closed interval $[0,1]$, as a result of the squashing property of the logistic function. The term $\frac{(\mathbf{y}+1)}{2}$ in equation (3.12) can be viewed as mapping the original $\{-1,1\}$ encoded output values into $\{0,1\}$, such that $\hat{\mathbf{y}}$ and $\frac{(\mathbf{y}+1)}{2}$ are guaranteed to lie in the same range.

Substituting equations (3.11) and (3.12) into (3.10), we get the iterative estimation of $\mathbf{w}_j$,

$$\mathbf{w}_j|_{new} = \left(\mathbf{X}^T \boldsymbol{\Lambda} \mathbf{R} \mathbf{X}\right)^{-1} \mathbf{X}^T \boldsymbol{\Lambda} \mathbf{R} \left[\mathbf{X}\mathbf{w}_j|_{old} - \mathbf{R}^{-1}\left(\hat{\mathbf{y}} - \frac{(\mathbf{y}+1)}{2} \right) \right] \tag{3.13}$$

Based on equation (3.13), the iterative estimation of $\mathbf{w}_j|_{new}$ can be viewed as the solution to a weighted least-squares problem with the *pseudo weights* $\boldsymbol{\Lambda}\mathbf{R}$ and the *pseudo targets* $\mathbf{X}\mathbf{w}_j|_{old} - \mathbf{R}^{-1}\left(\hat{\mathbf{y}} - \frac{\mathbf{y}+1}{2}\right)$. From our experiment results, the convergence of this IRLS estimation is quite fast in practice.

In equation (3.9), the type of $\boldsymbol{\Sigma}_j^{\mathbf{x}}|_{new}$ can have different choices, and the different choices essentially correspond to the models with different complexities. For example, simple models can use a *spherical* covariance $\boldsymbol{\Sigma}_j^{\mathbf{x}}|_{new} = \sigma^2\mathbf{I}$ or a diagonal covariance $\boldsymbol{\Sigma}_j^{\mathbf{x}}|_{new} = \mathrm{diag}\left[\sigma_1^2, \ldots, \sigma_D^2\right]$. More complex models can be obtained by introducing more parameters into the covariance matrix, e.g., by using an arbitrary *symmetric* and *positive semi-definite* matrix. Besides using different formulations of covariance matrices, the complexity of ENRBFC can also be controlled by tuning the number of the mixture components used in equation (3.4). The appropriate number of basis functions can be estimated by several model selection methods, such as the k-fold cross validation approach or the newly developed Bayesian Ying-Yang criteria [Hu and Xu 2004]. The results of ENRBFC with different types of covariance matrices are compared in the experiments in Section 3.3.

3.2 Comparisons of ENRBFC with Other Similar Classifiers

In this section we introduce some classifiers that have close connections to the ENRBFC model. These models are related to each other in the sense that all of them can be viewed as special applications of an underlying Gaussian Mixture Model (GMM) [McLachlan and Peel 2000; Nabney 2002; Bishop 2006].

Recall that based on equation (3.3), the marginal pdf of input $\mathbf{x}$ is essentially represented by a mixture of Gaussians,

$$\begin{aligned} p(\mathbf{x}) &= \sum_{j=1}^{M} p(j)\, p(\mathbf{x} \mid j) \\ &= \sum_{j=1}^{M} p(j)\, \mathrm{N}\left(\mathbf{x}; \boldsymbol{\mu}_j^{\mathbf{x}}, \boldsymbol{\Sigma}_j^{\mathbf{x}}\right) \end{aligned} \tag{3.14}$$

That is, ENRBFC is based on an underlying GMM in input space. From this point view, ENRBFC *softly* divides the input space into M local neighborhoods, and represents the global input-output mapping as a combination of the local models in each neighborhood, which gives $p(C_k \mid \mathbf{x}) = \sum_{j=1}^{M} p(j \mid \mathbf{x})\, p(C_k \mid \mathbf{x}, j)$. The other classifiers that will be discussed in this section resemble ENRBFC in the way that they all represent $p(\mathbf{x})$ based on a GMM. The difference lies in their different assumptions of the local models and how those models are trained by the corresponding learning algorithms.

3.2.1 The GMC Model

The Gaussian Mixture Classifier (GMC) can be viewed as a *generative* classification model in the sense that it models $p(\mathbf{x} \mid C_k)$ and $p(C_k)$ separately for each class label C_k. This is different from a *discriminative* model such as ENRBFC, which models the conditional output probability $p(C_k \mid \mathbf{x})$ directly. In a generative model, the prediction of $p(C_k \mid \mathbf{x})$ is derived by applying the *Bayesian theorem* to $p(\mathbf{x} \mid C_k)$ and $p(C_k)$ after they are estimated separately. And $p(C_k \mid \mathbf{x})$ is just the normalized version of the product $p(\mathbf{x} \mid C_k)\, p(C_k)$.

In GMC, a GMM is built separately in the input space for each class C_k, i.e.,

$$p(\mathbf{x} \mid C_k) = \sum_{j=1}^{M} p\left(\mathbf{x} \mid j, C_k, \boldsymbol{\theta}_{kj}\right) p(j \mid C_k) \tag{3.15}$$

Thus, the marginal input pdf $p(\mathbf{x})$ is itself a mixture of the Gaussian mixtures.

At the training stage, each $p(\mathbf{x}|C_k)$ is estimated by some unsupervised algorithms such as EM or the clustering algorithms. These estimations are separate for the training instances with different target outputs C_k. $p(C_k)$ can also be estimated by some simple and fast methods, such as by estimating the proportion of C_k with respect to the whole training sample. Essentially the estimations of both $p(\mathbf{x}|C_k)$ and $p(C_k)$ can be viewed as special cases of maximum likelihood estimation.

The main advantages of GMC are its easy implementation and its relatively fast training stage. Although it could suffer a decrease of accuracy compared to other supervised learning models, it is still able to give very good performances in practice given that there are enough mixture components used in the model.

3.2.2 The Common Basis GMC Model

From the last section we can see that the underlying assumption of GMC is to use separate GMMs for each class of data. This assumption brings GMC a lot of flexibilities in practice. However, if the data in one of the classes are scarce, fitting a separate model could lead to the over-fitting issue for that class. To overcome this, an alternative method is to use a common pool of mixture components for the whole dataset of all classes, i.e.,

$$p(\mathbf{x}) = \sum_{j=1}^{M} p\left(\mathbf{x} \mid j, \boldsymbol{\theta}_j\right) p(j) \text{ for } \forall\ C_k \qquad (3.16)$$

We call this model the *Common-Basis GMC* to distinguish it from the GMC model discussed in the last section. Based on equation (3.16), the predictions of this model are made by,

$$p(C_k|\mathbf{x}) = \sum_{j=1}^{M} p(j|\mathbf{x})\, p(C_k|\mathbf{x}, j) \qquad (3.17)$$

where $p(j\,|\,\mathbf{x})$ is derived based on equation (3.16) and $p(C_k\,|\,\mathbf{x}, j)$ is estimated as a constant value such as the proportion of C_k in the *j*th component. The common-basis GMC model is similiar to the NRBF model (in equation (2.15)) in the sense that both models divide the input space into some local neighborhoods that are delimited by the membership function $p(j\,|\,\mathbf{x})$, and represent their individual local estimations in each neighborhood as a constant-value model, i.e., $p(C_k\,|\,\mathbf{x}, j)$ for common-basis GMC or $E[y\,|\,\mathbf{x}, j]$ for NRBF. Their difference is that $E[y\,|\,\mathbf{x}, j]$ can be arbitrary real numbers and thus generally used in regressions, whereas $p(C_k\,|\,\mathbf{x}, j)$ must be a valid probability value and is usually used in classifications. Thus, the common-basis GMC model is an analogy to the NRBF model in a classification context.

The training of common-basis GMC can be either unsupervised or supervised. In the unsupervised method, the estimation of $p(\mathbf{x})$ in equation (3.16) is based on the input information alone (e.g., by clustering), and $p(C_k\,|\,\mathbf{x}, j)$ can be estimated as the proportion of C_k in the *j*th component. On the other hand, due to its similarity to NRBF, the common-basis GMC can also be trained by a supervised EM based training algorithm that is similar to the one in NRBF in Section 2.2.2, that is, the GMM parameters of $p(\mathbf{x})$ and $p(C_k\,|\,\mathbf{x}, j)$ are estimated iteratively by the interleaved expectation and maximization steps. Compared to the EM algorithm in NRBF, the major modification here is the different parameter updating at the maximization step, due to the different assumptions for the local models in GMM.

In general, the accuracy of the supervised method is expected to be better than the unsupervised one, considering the extra output information used in the training stage. However, the practical difference between them is usually not significant, especially when the number of basis functions used in the model is relatively larger than the number of classes. The results from both training algorithms will be compared in Section 3.3.

3.2.3 The Two Step NRBF Classifier

The traditional NRBF model trained by the fast two-step algorithm can also be used to do classifications, with the modification of adding an extra logistic/softmax output function to the output. Taking binary classifications as an example, the posterior probability of class label $C=1$ is given by,

$$\begin{aligned} p(C=1\,|\,\mathbf{x}) &= \sigma\left(f(\mathbf{x})\right) \\ &= \sigma\left(\sum_{j=1}^{M} w_j \tilde{\phi}_j\left(\mathbf{x}\,|\,\boldsymbol{\theta}_j\right)\right) \end{aligned} \tag{3.18}$$

where $\sigma(a)$ is the logistic function and $f(\mathbf{x})$ is the output of the regression NRBF model.

The training of this classifier is directly based on the two-step training algorithm for the regression NRBF model, with the exception that now the coefficients w_j cannot be directly estimated by solving a linear system at the second step. Here the optimization of w_j becomes nonlinear due to the presence of the logistic function. In general, a re-weighted least squares (IRLS) algorithm can be used here to estimate w_j in an iterative way.

From the above discussion, we can see that all these classifiers including ENRBFC, essentially take a *divide-and-conquer* approach by fitting GMMs in the input space and thus dividing it into several overlapped local neighborhoods. The boundaries of these neighborhoods are *soft* in the sense that the neighborhoods are represented by continuous-value probability functions. It is the number of these local neighborhoods that effectively controls the models' complexities. And their complexities can be estimated at the model selection stage by the *k*-fold cross validation process [Hastie, Tibshirani et al. 2001].

In spite of these similarities, the differences among these models are significant. For example, ENRBFC uses a logistic regression model (analogical to linear model in regression) to model its outputs, whereas the other models essentially use constant value modeling. These differences in structures directly lead to the different training methods for different models. Here we summarize the differences among these models into Table 3-2.

Table 3-2: Comparison of different GMM-based classifiers

Model	*Output Model Assumption*	*Training Algorithm*
ENRBFC	logistic regression	supervised
GMC	constant	unsupervised
Common-Basis GMC	constant	supervised/unsupervised
Two-Step NRBF Classifier	logistic of constant	hybrid (two step training)

Further experimental comparisons of these classifiers will be discussed in the next section.

3.3 Experimental Results and Analysis

This section composes of several regression and classification experiments, with the intention to compare the proposed ENRBFC model with their corresponding peers. These experiments help to reveal the advantages of ENRBFC in achieving better accuracies in some learning tasks. To avoid unpleasant domination of specific features and over-fitting, the same *normalization* and *cross-validation* strategies as described in Section 2.5 are also used here. The accuracy of models are measured by their *classification rates*.

The ENRBFC model is compared with several other classifiers that are discussed in Section 3.2, namely the GMC, the Common-Basis GMC and the two-step NRBF Classifier. As the

Common-Basis GMC can be trained by either an unsupervised or supervised algorithm, the results from both are presented. Moreover, as the types of covariance matrices also play an important role in models' prediction performance (discussed in Section 3.1.2), we will also present the results for different types of covariance matrices in most of the experiments.

3.3.1 Artificial Binary Classification

Starting with a simple two dimensional classification problem, we show the results of different models in an intuitive way by plotting their decision boundaries against the benchmark *Bayesian boundary*. The algorithm that generates the training data is cited from an example in the NETLAB toolbox [Nabney 2002]. The inputs consist of 200 two-dimensional vectors and are generated according to a mixture distribution of three Gaussians. The priors of these Gaussian components are 0.5, 0.25 and 0.25 respectively. And the means and covariance matrices of different components are [0, -0.1], [1, 1], [1, -1] and $\begin{bmatrix} 0.625 & -0.2165 \\ -0.2165 & 0.875 \end{bmatrix}$, $\begin{bmatrix} 0.2241 & -0.1368 \\ -0.1368 & 0.9759 \end{bmatrix}$, $\begin{bmatrix} 0.2375 & 0.1516 \\ 0.1516 & 0.4125 \end{bmatrix}$. The inputs are classified into two classes: the data generated by component 1 are categorized into class 1 and the other two components are categorized into class 2. Since the data are two dimensional and we know its real distribution a prior, it is very convenient to draw the results from different models in a 2-D plotting.

The classification boundaries from different models against the Bayesian optimal boundary are plotted in Figure 3-1 to Figure 3-5. In these figures, the dashed lines are boundaries generated by the individual models' predictions, and the solid line is the Bayesian boundary. The Bayesian boundary is calculated by applying the Bayesian theorem to the known distribution that generates the data. In all the learning models a *full* covariance matrix is specified, which is actually parameterized as an arbitrary symmetric and positive semi-definite matrix. We didn't try other shapes of covariance here because we already know its real shape a prior. In the results we also give the optimal model complexity (in terms of the number of GMM components or basis functions) used in the different classifiers, which are estimated by the three-fold cross validation. Although we know that the number of components generating the data is actually 3, the number of components used in different learning models is not necessarily the same, as this complexity is determined by both the data complexity and the relative approximating ability of the models together.

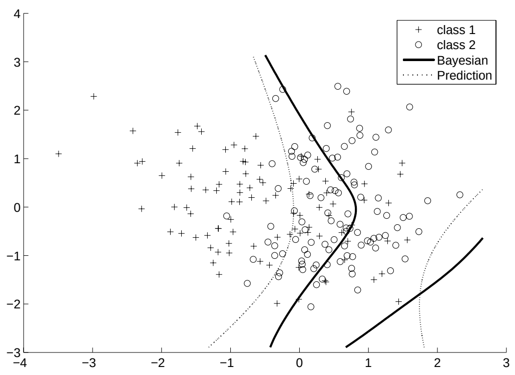


Figure 3-1: GMC decision boundary: optimal number of components is totally 2, one for each class

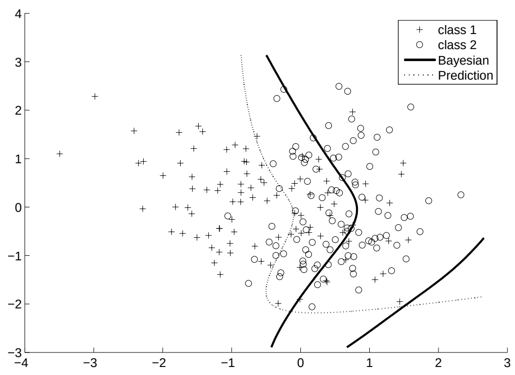


Figure 3-2: Unsupervised common-basis GMC decision boundary: optimal number of components is 3

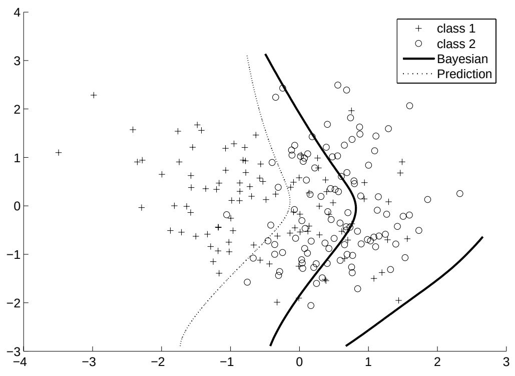


Figure 3-3: Supervised common-basis GMC decision boundary: optimal number of components is 3

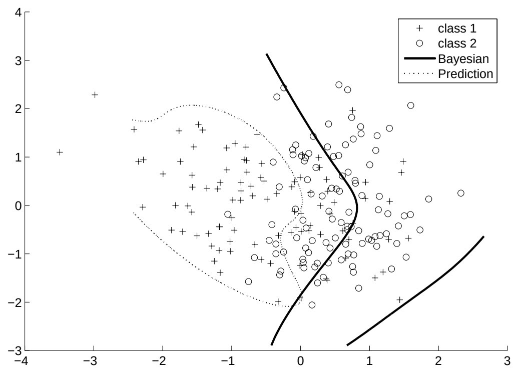


Figure 3-4: Two-step NRBF Classifier decision boundary: optimal number of basis functions is 4

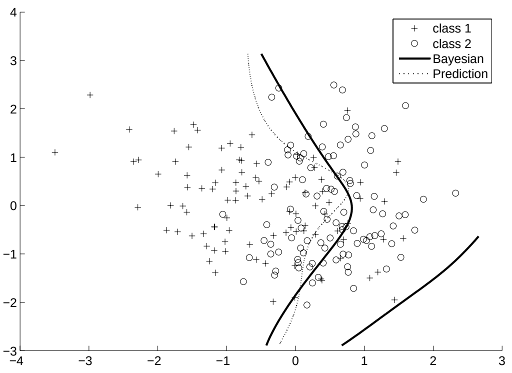


Figure 3-5: ENRBFC decision boundary: optimal number of basis functions is 3

Among the others, the decision boundary given by ENRBFC seems very close to the Bayesian boundary, especially in the vicinity of the overlapping area between the two classes. Correctly classifying instances in this vicinity poses a challenge for the classifiers, because of the asymmetry distribution in this area. Some local estimation based models intend to be biased in these areas as the results show in Figure 3-1~Figure 3-4. On the other hand, as we discussed in Section 2.4 and 3.2, ENRBFC is able to reduce this bias by using a linear fitting. As the common-basis GMC can be trained either in a supervised or unsupervised way, the results from both learning strategies are presented in Figure 3-2 and Figure 3-3 respectively. From the results we can see that the difference between these two training manners is not very significant due to the simplicity of this classification problem. As a general result, since the decision boundaries only depend on the *relative values* of the outputs, the differences between the supervised and unsupervised manners in classifications are usually not as significant as in regressions. This also explains why a lot of unsupervised or hybrid-mannered models are able to achieve the comparable performance as the supervised ones in classifications.

3.3.2 Wisconsin Breast Cancer Diagnosis

In the following two sections we will use the different classifiers to solve some problems from the real-world medical applications. In particular, the models are used to predict the breast cancer diagnosis result in this experiment. The data are from the Wisconsin Breast Cancer Diagnosis (WDBC) dataset which is online available at the Machine Learning Repository of University of California, Irvine (UCI), which is online available at http://www.cs.wisc.edu/~olvi/uwmp/cancer.html. The training set consists of the information of 569 diagnosed patients, among which 357 instances are classified as benign and the rest 212 instances as malignant. The inputs are 30 test results of the patients, extracted from a digitized image of a fine needle aspirate (FNA).

Table 3-3: Comparisons of different classifiers on WDBC data

	Accuracy		*# of Basis Functions*	*Training time*
GMC	spherical	91.57%	10	0.59 secs
	diagonal	95.60%	10	0.30 secs
	full	94.20%	10	0.47 secs
Unsupervised common-basis GMC	spherical	89.98%	5	0.25 secs
	diagonal	92.62%	9	0.52 secs
	full	94.02%	2	0.19 secs
Supervised common-basis GMC	spherical	88.40%	4	0.08 secs
	diagonal	92.09%	9	0.19 secs
	full	94.20%	2	0.19 secs
Two-step NRBF Classifier	spherical	62.74%	3	0.19 secs
	diagonal	69.40%	8	0.41 secs
	full	71.70%	9	0.99 secs
ENRBFC	spherical	96.66%	2	0.31 secs
	diagonal	98.59%	2	0.59 secs
	full	98.95%	2	0.64 secs

The results from different classifiers are compared in Table 3-3. For each model, the number of basis functions is determined by the three-fold cross validation. Besides, different *shapes* of covariance matrix are also compared in the experiment. Among the rest, ENRBFC beats the other models in accuracy whereas its training time remains at a moderate level. The GMC model also gives good accuracies at a cost of using more basis functions. Using a lot of basis functions could lead to the risk of over-fitting in practice, as it is common in real-world applications that the training data are relatively scarce compared to the complexity of the learning task. This is especially true in the biomedical applications where the occurrences of a specific class are extremely rare. In fact, the experiment in the next section is such a case. The performance of the two-step NRBF classifier is considered poor compared to the other models. This is not surprising because the two-step NRBF (in Section 3.2.3) is based on a direct adaptation of the corresponding regression model. As we have reiterated several times, this kind of direct adaptation from regression models to classifications will not be very successful in general, due to the intrinsic difference between classification and regression.

There are totally three types of covariance matrix experimented for each learning model, namely spherical, diagonal and full parameterization. Because the number of parameters in these specifications of matrices is different, the flexibility degrees of the resulted models are also different. In the most flexible specification (the full parameterized covariance matrix), a covariance can actually be an arbitrary symmetric and positive semi-definite matrix. Although in this experiment it seems that a more flexible matrix specification is able to give better accuracies, this is not generally true in practice. After all, the shape of the covariance should be determined by the intrinsic structure of the data to be modeled. The experiment in the next section will demonstrate this.

3.3.3 Ovarian Cancer Diagnosis

The last experiment used is also from a real-world medical application—the ovarian cancer diagnosis. The training data consist of 169 diagnosed instances collected from the patients' records over five years in the department of oncology and gynaecology of National University Hosptial (NUH), Singapore. The inputs are 27 features from the patients' test results, including blood platelet, thrombelastography time, plasminogen activators and etc. In accordance with the staging system suggested by the international federation of gynaecology and obstetrics, the patients are classified into five classes, namely *normal*, *benign*, *borderline*, *stages I/II*, and *stages III/IV*.

The learning problem in this experiment appears more difficult than the previous one as there are more than two classes involved. In other words, it is a multi-classification problem. As a consequence, the data to be modeled and the relations among different classes are more complicated. Another extra difficulty in the current problem arises from the relatively small training data set compared to its high dimensional space. The input dimensionality in this experiment is about the same as the previous experiment whereas the training data size is much smaller. All these difficulties make the learning models more vulnerable to the curse of dimensionality. To do the multi-classification, we modify ENRBFC and the NRBF classifier by replacing the logistic function with the softmax function, such that the sum-to-one constraint of different output values is still assured. As for GMC and the common-basis GMC, the multi-classification is almost as the same as the binary classification.

Table 3-4: Comparisons of different classifiers on OVA data

	Accuracy		*# of Basis Functions*	*Training time*
GMC	spherical	64.50%	15	0.55 secs
	diagonal	68.64%	15	0.50 secs
	full	67.46%	5	0.06 secs
Unsupervised common-basis GMC	spherical	62.72%	5	0.13 secs
	diagonal	69.82%	8	0.23 secs
	full	66.27%	7	0.27 secs
Supervised common-basis GMC	spherical	63.91%	8	0.20 secs
	diagonal	69.23%	6	0.13 secs
	full	66.86%	6	0.44 secs
Two-step NRBF Classifier	spherical	53.25%	4	0.11 secs
	diagonal	59.76%	6	0.19 secs
	full	50.30%	6	0.23 secs
ENRBFC	spherical	94.08%	6	0.95 secs
	diagonal	82.25%	1	0.06 secs
	full	82.25%	1	0.06 secs

The comparison results are summarized in Table 3-4. As we have discussed, here the main challenge is the relative scarcity of the data with respect to the high input dimensionality. As a consequence, there is a “complexity dilemma” for each learning model: using fewer basis functions could result in the lack of enough approximating flexibility, whereas using more of them could lead to the over-fitting caused by the curse of dimensionality. This dilemma essentially reflects the intrinsic limitations of the local constant models learning in high dimensional space. The relatively poor accuracies of these models demonstrate our point. On the other hand, the ENRBFC model is more robust in this situation due to its extra flexibility brought by its structure assumption (in Section 3.2). Actually, it is able to give much better performance in the experiment, as we can tell from Table 3-4.

Please also note that a full parameterization covariance doesn't necessarily give the best performance, as we have discussed in the last section. In fact, the higher accuracies are usually achieved by using a spherical or diagonal matrix, because in a high dimensional space a fully parameterized covariance will introduce too many parameters, which in turn makes the models over-fit.

3.4 Summary

In this section, inspired by the ENRBF model, we proposed the ENRBFC model to handle the difficulties in high dimensional classification problems. Some traditional models usually have troubles with classification problems that have tens of input features and relatively scarce training instances. These problems are usually considered hard classification problems in the literature. Compared to these traditional models, ENRBFC is able to significantly improve learning accuracies and alleviate explosions of basis functions. Besides, its EM-based learning algorithm makes its training and evaluation stages both easy-to-implement and relatively fast. As a result, the ENRBFC model fits very well in these situations. The experiment results support this claim.

In the next chapter we will turn our attention to the computation issues of nonparametric models and discuss in detail one of the representatives—the Gaussian Process model.

Chapter 4

4 A Review of GP Model

Having examined the main computation issues of *parametric* models and proposed our solution—the ENRBFC model, in this chapter we turn to discussing *nonparametric* models. Although the main issues of parametric models mostly arise from the curse of dimensionality, the computation problems of nonparametric models are usually caused by their high computation costs for large-scale datasets. In this chapter, we explore in detail these computation issues by focusing on a particular nonparametric model—*Gaussian Process* (GP) [Williams and Rasmussen 1996; Rasmussen and Williams 2006]. We also check the various extensions of the traditional GP model that are proposed to reduce their computation costs in large-scale applications. These more efficient GP models are usually called *sparse* GPs in the literature in contrast to the traditional *full* GP model. The discussion of this chapter forms the basis of our research work in Chapter 5.

4.1 Gaussian Process Model

The *Gaussian Process* (GP) model has been well developed as a machine learning method since its first applied to regression problems [Williams and Rasmussen 1996]. As a typical nonparametric model, the GP model has several advantages over parametric ones such as Generalized Additive Model [Hastie and Tibshirani 1990], Neural Networks [Bishop 1995] and Radial Basis Function [Powell 1987].

The first advantage of GP models is that their model complexities are able to grow with training data in a natural way. Thus, GP models don't have the *under-fitting* problem that most parametric models usually suffer when the data to be modeled are too complex. The nature of this kind of flexibility in GP is best explained by *Mercer's theorem* [Konig 1986], which states that a non-degenerated GP model can actually be represented by an infinite sum of basis functions. On the other hand, the invited computations of these basis functions are still finite, according to an appealing result called *kernel trick* [Rasmussen and Williams 2006].

The second advantage of GP models is that their prediction results are usually more expressive than parametric models. As discussed in Section 2.1, many traditional learning models represent their predictions as point estimates. For example, their point estimates for regression problems are usually the conditional means of outputs, and the point estimates for classification problems are the maximum a posterior (MAP) probabilities of class labels. On the other hand, GP models are full Bayesian models and able to make predictions directly on the calculated posterior probabilities of outputs. In other words, the results of GP models don't have to be converted to point estimates by discarding the stochastic part of learning results. This usually leaves us a lot of choices in making decisions, especially when the point estimates end up in a tie. In fact, the point estimates from traditional models can be viewed as the special cases of GPs' Bayesian results.

Moreover, compared to parametric models, GP models have a more efficient way of doing model selection. To choose the appropriate model complexity, traditional models usually fit a group of related models with various complexities at the training stage, and then use a separate cross validation process to do model selection. This requires the training algorithm to be performed several times interleaved with the model selection process, resulting in a cost of dense computation. Conversely, GP models are able to perform both the training and model selection stages seamlessly in the same process. Through maximizing a criterion called *marginal likelihood* (or *evidence*), the *hyper-parameters* that control GPs' complexities can be viewed as just the ordinary parameters and thus be optimized the same way at the training stage. In practice this saves a lot of computation efforts and avoids the multiple interleaving runs of the training and model selection processes.

In the rest of this section we cover the basics of GP models and discuss their practical difficulties on large-scale applications. We will also discuss some remedies that have been proposed in the literature in the next section.

4.1.1 GP Regression Model

As discussed in Section 1.3, intuitively a regression problem can be viewed as the estimation of the underlying input-output function $f(\mathbf{x})$ based on a set of observations $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$. There are two main facts making this estimation a challenge. First, the observations $(\mathbf{x}_i, y_i)$ are usually bearing some interrupted noise, which requires learning models to provide some mechanisms to model this kind of uncertainty. Even if the observations were noise free, the problem would still be *ill-posed* as there are infinite number of candidate models that can equivalently fit the observations [Vapnik 1995]. This is just the reflection of the inductive nature of learning problems. The traditional way of approaching these problems is to put some extra constraints on learning models and thus control their complexities. During the training and model selection stage, the model with the lowest complexity that is capable of fitting the training data will be chosen, which we believe will give the best generalization accuracy on new instances. In the literature this principle is usually referred to as *Ockham's razor* [Bishop 2006]. GP models have a more convenient way to handle the learning challenge—instead of picking one model from others, they keep a whole set of models under consideration and make predictions by combining the results from them all. More importantly, the number of models under consideration could be infinite, which provides GP models essentially the infinite flexibility in modeling arbitrary datasets. In the follows we will review the underlying assumptions of GP models and how they handle the regression problems. GP models for classifications will be covered in Section 4.1.2.

By definition a *Gaussian Process* is a collection of random variables, any finite number of which has a joint Gaussian distribution [MacKay 2003; Rasmussen and Williams 2006]. A GP can be completely determined by two functions, namely the *mean function* $m(\mathbf{x}) = E\left[f(\mathbf{x})\right]$, and the *covariance function* (or *kernel function*) $k(\mathbf{x},\mathbf{x}') = E\left[\left(f(\mathbf{x}) - m(\mathbf{x})\right)\left(f(\mathbf{x}') - m(\mathbf{x}')\right)\right]$. In practice, we usually assume $m(\mathbf{x}) = 0$ without a loss of generality. This corresponds to assuming that the data to be modeled have a zero mean as we can always standardize the data first. Thus, it is the kernel function $k(\mathbf{x},\mathbf{x}')$ that makes different GP models behave differently.

In principle the only requirements a valid kernel function is to be *symmetric* and *positive semi-definite*, i.e., a kernel function $k(\mathbf{x},\mathbf{x}')$ always generates *symmetric* and *positive semi-definite* covariance matrices on arbitrary data. In practice however, computation convenience is also an important consideration in choosing a kernel function. Popular kernel functions include the *squared exponential isotropic* (SE-ISO) kernel, the *Matern* kernel, the *automatic relevance determination* (ARD) kernel and others. Among them, the ARD kernel is of special interest to us because it has a natural connection to feature selection and dimension reduction, which will be discussed as follows. Generally the ARD kernel can be represented as,

$$k_{ARD}(\mathbf{x},\mathbf{x}') = a^2 \exp\left[-\frac{1}{2}\sum_{d=1}^{D}\left(\frac{x_d - x'_d}{\lambda_d}\right)^2\right] \tag{4.1}$$

where $\mathbf{x}$ and $\mathbf{x}'$ are two input vectors and x_d is the *d*th component of $\mathbf{x}$. An ARD kernel is parameterized by a^2 and $\lambda_1,...,\lambda_D$, where a^2 is the *signal attitude* and λ_d is the *length-scale* on the *d*th dimension. Because a kernel function is directly related to the covariance of a stochastic process, a^2 essentially controls the variance of the GP signal, and λ_d controls how fast the signal varies along each dimension. In other words, λ_d can be viewed as a measure of the signal's smoothness on the *d*th input dimension. A big value of λ_d indicates the underlying signals change slowly with the input component x_d and vice versa. When λ_d is estimated at the training stage, a big estimation value indicates the output doesn't change much along x_d, and thus x_d could be marked as a non-significant feature. This way the estimation of an ARD kernel can be directly related to a feature selection process.

After choosing an appropriate kernel function, applying GP models to regression problems is straightforward. The underlying input-output function $f(\mathbf{x})$ can be modeled as a Gaussian Process, in which any finite number of instances $\{f(\mathbf{x}_1),..., f(\mathbf{x}_k)\}$ will form a Gaussian distribution. Following the conventions, we denote the training inputs (design matrix) as $\mathbf{X} = \begin{pmatrix} x_{11},...,x_{1D} \\ \vdots \\ x_{N1},...,x_{ND} \end{pmatrix}$ and outputs as $\mathbf{y} = [y_1,...y_N]^T$, where D is the input dimensionality and N is the number of training instances. The test inputs are denoted as matrix $\mathbf{X}_*$ and model's predictions for them are $\mathbf{f}_*$. To model an additive noise in the observations of outputs, we assume,

$$p\left(\mathbf{y}\,|\,\mathbf{f}\right) = \mathrm{N}\left(\mathbf{y};\mathbf{f},\sigma^2\mathbf{I}\right) \tag{4.2}$$

which corresponds to assuming an additive Gaussian noise ε with $E\,[\varepsilon] = 0$ and $E\left[\varepsilon^2\right] = \sigma^2$ in the output. From (4.2) we can see the value $\mathbf{f}$ is essentially the mean of outputs $\mathbf{y}$, i.e., the regression function.

Since $f\left(\mathbf{x}\right)$ is a Gaussian process, by definition any instances generated from it will follow a Gaussian distribution. Specially, the joint *prior* distribution of training and test outputs follows a Gaussian distribution,

$$p\left(\mathbf{f},\mathbf{f}_*\right) = \mathrm{N}\left(\mathbf{0},\begin{bmatrix}\mathbf{K}_{ff} & \mathbf{K}_{f*}\\ \mathbf{K}_{*f} & \mathbf{K}_{**}\end{bmatrix}\right) \tag{4.3}$$

Here we assume this Gaussian process has a zero mean function and $\mathbf{K}_{ff}$, $\mathbf{K}_{f*}$ and $\mathbf{K}_{**}$ are the respective parts of the covariance matrices generated by the kernel function. In details, $\mathbf{K}_{ff} = k\left(\mathbf{X},\mathbf{X}\right)$, $\mathbf{K}_{f*} = k\left(\mathbf{X},\mathbf{X}_*\right)$, and $\mathbf{K}_{**} = k\left(\mathbf{X}_*,\mathbf{X}_*\right)$.

In equations (4.2) and (4.3) we relate the underlying GP model to training outputs and new predictions respectively. Based on these equations, we can also work out the relation between training observations and predictions. This is done by combining equations (4.2) and (4.3) as

$$\begin{aligned} p\left(\mathbf{y},\mathbf{f}_*\right) &= \int p\left(\mathbf{y}\,|\,\mathbf{f}_*,\mathbf{f}\right) p\left(\mathbf{f}_*\,|\,\mathbf{f}\right) p\left(\mathbf{f}\right) d\mathbf{f} \\ &= \int p\left(\mathbf{y}\,|\,\mathbf{f}_*,\mathbf{f}\right) p\left(\mathbf{f}_*,\mathbf{f}\right) d\mathbf{f} \\ &= \int p\left(\mathbf{y}\,|\,\mathbf{f}\right) p\left(\mathbf{f}_*,\mathbf{f}\right) d\mathbf{f} \\ &= \mathrm{N}\left(\mathbf{0},\begin{bmatrix}\mathbf{K}_{ff}+\sigma^2\mathbf{I} & \mathbf{K}_{f*}\\ \mathbf{K}_{*f} & \mathbf{K}_{**}+\sigma^2\mathbf{I}\end{bmatrix}\right) \end{aligned} \tag{4.4}$$

Here in the equation we used the fact $p(\mathbf{y} \mid \mathbf{f}_*, \mathbf{f}) = p(\mathbf{y} \mid \mathbf{f})$, i.e., conditioning on the underlying training outputs $\mathbf{f}$, the observations $\mathbf{y}$ are independent of $\mathbf{f}_*$. This is a rational assumption because the observations $\mathbf{y}$ are just the underlying outputs $\mathbf{f}$ interrupted by noise. The derivation of the last step of equation (4.4) uses equation (5) in the Appendix.

Given the representation of the joint probability in equation (4.4), predicting new outputs is equivalent to finding out the posterior probability of $\mathbf{f}_*$, i.e.,

$$p(\mathbf{f}_* \mid \mathbf{y}) = \mathrm{N}\left(\bar{\mathbf{f}}_*, \Sigma_*\right) \tag{4.5}$$

where

$$\bar{\mathbf{f}}_* = \mathbf{K}_{*f}\left(\mathbf{K}_{ff} + \sigma^2\mathbf{I}\right)^{-1}\mathbf{y} \tag{4.6}$$

$$\Sigma_* = \mathbf{K}_{**} - \mathbf{K}_{*f}\left(\mathbf{K}_{ff} + \sigma^2\mathbf{I}\right)^{-1}\mathbf{K}_{f*} \tag{4.7}$$

This derivation is based on equation (3) in appendix. Equations (4.5) ~ (4.7) suggests that given the observations $\mathbf{y}$, the predictions $\mathbf{f}_*$ for the new inputs $\mathbf{X}_*$ ($\mathbf{X}_*$ is involved in the calculation of both $\mathbf{K}_{**}$ and $\mathbf{K}_{f*}$) also follows a Gaussian distribution. If the determinant of the prediction covariance Σ_* is small, a point estimate such as the prediction mean $\bar{\mathbf{f}}_*$ will be good enough in practice. This is exactly the point estimations used in traditional non-Bayesian methods. We can also see that Gaussian processes are nonparametric models as their prediction $\bar{\mathbf{f}}_*$ in equation (4.6) is represented by a weighted sum of the observations $\mathbf{y}$. If the number of observations increases, the number of items in this sum increases as well.

The predictions in equation (4.5) need the calculation of some covariance matrices $\mathbf{K}_{**}, \mathbf{K}_{f*}$ and $\mathbf{K}_{*f}$. These covariance matrices are generated by applying a specific kernel function to the corresponding data and thus parameterized by some hyper-parameters, e.g., the *signal attitude* and the *length-scales* in an ARD kernel. Similar to the hyper-parameters in parametric models, here these parameters also control the properties and thus the effective complexity of the underlying GP model. For GP models, besides the traditional model selection methods such as CV, there is an alternative way to optimize these hyper-parameters that is called *marginalization* in the literature. In details, the parameters $\boldsymbol{\theta}$ in a kernel function $k(\mathbf{x}, \mathbf{x}' \mid \boldsymbol{\theta})$ is optimized by maximizing the *marginal likelihood* of the training data,

$$\begin{aligned} p(\mathbf{y} \mid \boldsymbol{\theta}) &= \int p(\mathbf{y} \mid \mathbf{f})\, p(\mathbf{f})\, d\mathbf{f} \\ &= \mathrm{N}\left(\mathbf{y} \mid \mathbf{0}, \mathbf{K}_{\boldsymbol{\theta}} + \sigma^2 \mathbf{I}\right) \end{aligned} \tag{4.8}$$

where $\mathbf{K}_{\boldsymbol{\theta}}$ is the kernel matrix of the training inputs and σ^2 is the noise variance estimated from the data. By marginalizing out $\mathbf{f}$, the marginal likelihood $p(\mathbf{y} \mid \boldsymbol{\theta})$ only depends on the training data $\mathbf{y}$ and the hyper-parameter $\boldsymbol{\theta}$. And it turns out that a big value of $p(\mathbf{y} \mid \boldsymbol{\theta})$ favors a simple model over a complex one given both of them are able to model the data correctly. Thus, determining the kernel parameters in GP just corresponds to the model selection in a traditional model. More discussion about the rationality of $p(\mathbf{y} \mid \boldsymbol{\theta})$ as a model selection method can be found in [MacKay 2003; Bishop 2006].

In the above computations, the major overhead of a GP model lies in the inversion calculation $\left(\mathbf{K}_{ff} + \sigma^2 \mathbf{I}\right)^{-1}$ in equation (4.6) and (4.7). By using *Cholesky decomposition* [Press, A. et al. 1992], this inversion can be generally done in $\mathrm{O}\left(N^3\right)$. If a point estimate $\bar{\mathbf{f}}_*$

is sufficient, the time complexity of making predictions can be reduced to $O(N)$ by pre-computing the $\left(\mathbf{K}_{ff}+\sigma^2\mathbf{I}\right)^{-1}\mathbf{y}$ part that is independent of the new instances. However, the calculation of Σ_* in equation (4.7) still needs $O\left(N^2\right)$ time.

4.1.2 GP Classification Model

As we have discussed in Chapter 1, the only difference between a regression and a classification problem is their different output types. This difference makes the GP models for regression and classification problems very different. In this section we review the main GP classification frameworks.

As for classification problems, the biggest challenge is that the underlying output values in a classification problem cannot be directly modeled as a GP, since their values are not real numbers anymore. Thus, our basic assumption for regression cases (in equation (4.3)) doesn't hold for classifications. To reflect this change, a different conditional probability of output $\mathbf{y}$ is usually assumed in classifications,

$$\begin{aligned} p(\mathbf{y}=1\,|\,\mathbf{f}) &= \sigma(\mathbf{f}) \\ p(\mathbf{y}=-1\,|\,\mathbf{f}) &= 1-\sigma(\mathbf{f}) \end{aligned} \tag{4.9}$$

where $\mathbf{y}$ is the observed outputs and $\mathbf{f}$ is the underlying output values we are going to model. The function $\sigma(\cdot)$ is usually called *squashing* function because it takes real numbers as inputs and generates outputs in the range $[0,1]$, as if the function squashes the real numbers to that range. Popular choices of $\sigma(\cdot)$ include the *logistic* function $s(x)=e^x\big/\left(1+e^x\right)$ and the *probit* function $\Phi(x)=\int_{-\infty}^{x}\mathrm{N}\left(a\,|\,0,1\right)\mathrm{d}a$ .It is interesting to compare the two different assumptions in equation (4.2) and (4.9) that correspond to regressions and classifications respectively. In fact, with the assistance of equation (4.9), we can model the underlying value $\mathbf{f}$ as a Gaussian process again as its values are real numbers now. That is, equation (4.3) is also a valid assumption to classifications under the assumption of equation (4.9).

However, because the probability $p(\mathbf{y}\,|\,\mathbf{f})$ in (4.9) is not a Gaussian anymore, the probabilities $p(\mathbf{y},\mathbf{f}_*)$ (in equation (4.4)) and $p(\mathbf{f}_*\,|\,\mathbf{y})$ (in equation (4.5)) are not Gaussian either. In fact, without approximations, it is hard to find the close-form representations for these quantities. In the literature, there have been several methods developed to approach this problem. Generally they can be categorized into two types—the *sampling* methods and the *analytical* methods. In sampling methods such as *Markov chain Monte Carlo* [Neal 1999], these distributions (e.g., $p(\mathbf{f}_*\,|\,\mathbf{y})$) are estimated in an asymptotic way in which large data samples are generated and the estimations are made on them. On the other hand, the analytical methods such as *Laplace approximation* [Rasmussen and Williams 2006], *Expectation propagation* [Minka 2001], and *variational methods* [Gibbs and MacKay 2000], develop the analytic representations by approximating these non-Gaussian distributions with the Gaussian ones. These approximations will enable us to apply the evaluations and inferences that are similar to GP regression models. More details can be found in [Rasmussen and Williams 2006]. Here we review the *Laplace approximation* framework in the following sections, as it forms the discussion basis of our research work in the next chapter.

4.1.3 Laplace Approximation

As discussed in the previous sections, making predictions for regressions in GP corresponds to calculating the posterior distribution $p(\mathbf{f}_*\,|\,\mathbf{y})$, which can be explicitly expressed as,

$$p(\mathbf{f}_*\,|\,\mathbf{y}) = \int p(\mathbf{f}_*\,|\,\mathbf{f})\,p(\mathbf{f}\,|\,\mathbf{y})\,d\mathbf{f} \tag{4.10}$$

For regression cases, we developed the close-form for $p(\mathbf{f}_* \mid \mathbf{y})$ through equation (4.5) to (4.7). These calculations are straightforward as both $p(\mathbf{f}_* \mid \mathbf{f})$ and $p(\mathbf{f} \mid \mathbf{y})$ are Gaussian.

As for classifications, since our output values are not real numbers anymore, we made predictions through two steps. Besides equation (4.9), an extra step is needed in classification, i.e.,

$$\begin{aligned} p(\mathbf{y}_* = 1 \mid \mathbf{y}) &= \int p(\mathbf{y}_* = 1 \mid \mathbf{f}_*)\, p(\mathbf{f}_* \mid \mathbf{y})\, d\mathbf{f}_* \\ &= \int \sigma(\mathbf{f}_*)\, p(\mathbf{f}_* \mid \mathbf{y})\, d\mathbf{f}_* \end{aligned} \tag{4.11}$$

Where $\sigma(\cdot)$ is the *squashing* function and $p(\mathbf{f}_* \mid \mathbf{y})$ can be derived from equation (4.10). The main difficulty that prevents GP regression framework from being directly applied to classifications is that the quantity $p(\mathbf{f}_* \mid \mathbf{y})$ in equation (4.10) is not a Gaussian. As a consequence, this makes the calculation of both $p(\mathbf{f}_* \mid \mathbf{y})$ and $p(\mathbf{y}_* = 1 \mid \mathbf{y})$ in equation (4.11) analytically intractable.

The Laplace approximation framework tackles this calculation difficulty by utilizing a Gaussian approximation $q(\mathbf{f} \mid \mathbf{y})$ to the non-Gaussian posterior $p(\mathbf{f} \mid \mathbf{y})$ in the integral (4.10). In this way, the non-Gaussian $p(\mathbf{f}_* \mid \mathbf{y})$ can also be approximated by a Gaussian distribution $q(\mathbf{f}_* \mid \mathbf{y})$. After this approximation the calculation of predictions in equation (4.11) will be straightforward as $p(\mathbf{y}_* = 1 \mid \mathbf{y})$ is just the expectation of function $\sigma(\mathbf{f}_*)$ over a Gaussian distribution $q(\mathbf{f}_* \mid \mathbf{y})$.

To approximate $p(\mathbf{f} \mid \mathbf{y})$ by a Gaussian $q(\mathbf{f} \mid \mathbf{y})$, the Laplace approximation uses a second order Taylor expansion of $\log p(\mathbf{f} \mid \mathbf{y})$, i.e.,

$$q(\mathbf{f} \mid \mathbf{y}) = \mathrm{N}\left(\mathbf{f} \mid \hat{\mathbf{f}}, \mathbf{A}^{-1}\right) \tag{4.12}$$

Where

$$\begin{aligned} \hat{\mathbf{f}} &= \arg\max_{\mathbf{f}}\, p(\mathbf{f} \mid \mathbf{y}) \\ \mathbf{A} &= -\nabla\nabla \log p(\mathbf{f} \mid \mathbf{y})\big|_{\mathbf{f}=\hat{\mathbf{f}}} \end{aligned} \tag{4.13}$$

From the above equation we can see that $\hat{\mathbf{f}}$ is the point at the maximum of the posterior and $\mathbf{A}$ is the Hessian of the negative log posterior at $\hat{\mathbf{f}}$.

We can find $\hat{\mathbf{f}}$ and $\mathbf{A}$ by maximizing $\log p(\mathbf{f} \mid \mathbf{y})$ with respect to $\mathbf{f}$. The quantity $\log p(\mathbf{f} \mid \mathbf{y})$ is proportional to $\log p(\mathbf{y} \mid \mathbf{f})\, p(\mathbf{f})$ considering $p(\mathbf{f} \mid \mathbf{y}) = p(\mathbf{y} \mid \mathbf{f})\, p(\mathbf{f}) / p(\mathbf{y})$ and $p(\mathbf{y})$ is independent of $\mathbf{f}$. Thus, the only part that needs to be considered in the maximization is the un-normalized posterior, which can be expressed as,

$$\begin{aligned} \Psi(\mathbf{f}) &\triangleq \log p(\mathbf{y} \mid \mathbf{f}) + \log p(\mathbf{f}) \\ &= \log p(\mathbf{y} \mid \mathbf{f}) - \frac{1}{2}\mathbf{f}^T \mathbf{K}_{ff}^{-1}\mathbf{f} - \frac{1}{2}\log\left|\mathbf{K}_{ff}\right| - \frac{N}{2}\log 2\pi \end{aligned} \tag{4.14}$$

Here we use the fact that $p(\mathbf{f})$ is a Gaussian distribution with kernel $\mathbf{K}_{ff}$ as described in equation (4.3). Differentiating equation (4.14) with respect to $\mathbf{f}$ we obtain,

$$\begin{aligned} \nabla\Psi(\mathbf{f}) &= \nabla \log p(\mathbf{y} \mid \mathbf{f}) - \mathbf{K}_{ff}^{-1}\mathbf{f} \\ \nabla\nabla\Psi(\mathbf{f}) &= \nabla\nabla \log p(\mathbf{y} \mid \mathbf{f}) - \mathbf{K}_{ff}^{-1} \end{aligned} \tag{4.15}$$

With equation (4.14) and (4.15), $\mathbf{A}$ in equation (4.13) can be calculated as,

$$\begin{aligned} \mathbf{A} &= -\nabla\nabla\Psi(\mathbf{f})\big|_{\mathbf{f}=\hat{\mathbf{f}}} \\ &= \mathbf{K}_{ff}^{-1} - \nabla\nabla \log p\left(\mathbf{y} \mid \hat{\mathbf{f}}\right) \end{aligned} \tag{4.16}$$

As for $\hat{\mathbf{f}}$ in equation (4.13), it corresponds to the point that maximizes $\Psi(\mathbf{f})$, so can be found by Newton's method using the iterative step,

$$\begin{aligned}\hat{\mathbf{f}}|_{new} &= \hat{\mathbf{f}} - \left(\nabla\nabla\Psi\left(\hat{\mathbf{f}}\right)\right)^{-1}\nabla\Psi\left(\hat{\mathbf{f}}\right) \\ &= \hat{\mathbf{f}} + \left(\mathbf{K}_{ff}^{-1} - \nabla\nabla\log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right)\right)^{-1}\left(\nabla\log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right) - \mathbf{K}_{ff}^{-1}\hat{\mathbf{f}}\right) \\ &= \left(\mathbf{K}_{ff}^{-1} - \nabla\nabla\log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right)\right)^{-1}\left(\nabla\log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right) - \nabla\nabla\log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right)\hat{\mathbf{f}}\right)\end{aligned} \tag{4.17}$$

In equation (4.16) and (4.17), the derivations of $\nabla\log p(\mathbf{y}\,|\,\mathbf{f})$ and $\nabla\nabla\log p(\mathbf{y}\,|\,\mathbf{f})$ will depend on the specific squashing function used in (4.11). For example, for a logistic function, the two quantities can be derived as

$$\begin{aligned}\nabla\log p(\mathbf{y}\,|\,\mathbf{f}) &= \mathbf{t} - \boldsymbol{\pi} \\ \nabla\nabla\log p(\mathbf{y}\,|\,\mathbf{f}) &= -\boldsymbol{\pi}(1-\boldsymbol{\pi})\end{aligned} \tag{4.18}$$

where $\boldsymbol{\pi} = p(\mathbf{y}=1\,|\,\mathbf{f})$ and $\mathbf{t} = (\mathbf{y}+1)/2$ given $\mathbf{y}$ is encoded by -1 or $+1$. Substituting equation (4.16) and (4.17) back into (4.12), we can get the approximation $q(\mathbf{f}\,|\,\mathbf{y})$. And the predictions for new instances will be made by equation (4.11). More details can be found in [Rasmussen and Williams 2006].

4.1.4 Marginal Likelihood and Hyperparameters Tuning

As discussed in Section 4.1.1, GP models are parameterized by hyperparameters such as the parameters in the kernel functions. The value of these hyperparameters has a big influence on the complexity and the generalization performance of GP models. Thus, optimizing these hyperparameters is usually viewed as equivalent to the model selection step for traditional methods [MacKay 1998; Bishop 2006; Rasmussen and Williams 2006; Snelson 2007]. In the literature different hyperparameter optimization methods have been developed for Gaussian process. For example, the cross validation method (introduced in Chapter 2) that is widely used for traditional models can also be used to tune hyperparameters in GP models [Wahba 1990; Rasmussen and Williams 2006].

However, one of the main criticisms for using CV with GP models arises from the fact that a GP model usually has a vector of multiple hyperparameters. For example, there is one scale length parameter for each input dimension in the ARD kernel introduced in Equation (4.1). In such a situation, exploring combinations of settings for such parameters could, in the worst case, require a number of training runs that is exponential in the number of parameters [Bishop 2006]. As a consequence, this can prove problematic in practice since the computations of GP itself are computationally expensive. The same conclusion has also been given by the others such as in [Rasmussen and Williams 2006].

Fortunately, there is an alternative to the cross validation method. The Bayesian inference principle underlying GP models provide a persuasive and consistent framework to tune the hyperparameters in GP models. Under this framework, the hyperparameters are optimized by maximizing the *marginal likelihood* that is introduced in Equation (4.8). Compared to CV, the main advantage of this framework is that it relies only on the training data and allows multiple hyperparameters to be tuned in a single training run. More detail about the rationality of the marginal likelihood method can be found in [Bishop 2006; Rasmussen and Williams 2006]. However, to use this method, the derivatives of the marginal likelihood need to be computed. Although these computations for GP regression models are analytically tractable, the same cannot be said with classification frameworks such as the Laplace approximation. In the rest of this section, we will briefly deduct the derivatives of the marginal likelihood for Laplace approximation, which forms the discussion foundation for our research work in Chapter 5.

Recall that in Section 4.1.1 we represent the marginal likelihood in equation (4.8). For regressions, the computation of both the marginal likelihood and its derivatives with respect to the hyperparameters $\boldsymbol{\theta}$ are straightforward as the distributions in the integral are all Gaussian. But in a classification case such as in the Laplace approximation discussed in the last section, both computations are analytically intractable due to the non-Gaussian nature of $p(\mathbf{y} \mid \mathbf{f})$ in the integral.

To compute the marginal likelihood in the Laplace approximation, we substitute equation (4.14) into equation (4.8) and obtain,

$$\begin{aligned} p(\mathbf{y} \mid \boldsymbol{\theta}) &= \int p(\mathbf{y} \mid \mathbf{f}, \boldsymbol{\theta})\, p(\mathbf{f} \mid \boldsymbol{\theta})\, d\mathbf{f} \\ &= \int \exp\left(\Psi(\mathbf{f})\right) d\mathbf{f} \end{aligned} \tag{4.19}$$

Here $\Psi(\mathbf{f})$ is also a function of $\boldsymbol{\theta}$ as $\mathbf{f}$ depends on $\boldsymbol{\theta}$. Using a second order Taylor expansion of $\Psi(\mathbf{f})$ around $\hat{\mathbf{f}}$, i.e., $\Psi(\mathbf{f}) \simeq \Psi(\hat{\mathbf{f}}) - \frac{1}{2}(\mathbf{f} - \hat{\mathbf{f}})^T \mathbf{A}(\mathbf{f} - \hat{\mathbf{f}})$, we obtain an approximation $q(\mathbf{y} \mid \boldsymbol{\theta})$ to the marginal likelihood as,

$$p(\mathbf{y} \mid \boldsymbol{\theta}) \simeq q(\mathbf{y} \mid \boldsymbol{\theta}) = \exp\left(\Psi(\hat{\mathbf{f}})\right) \int \exp\left(-\frac{1}{2}(\mathbf{f} - \hat{\mathbf{f}})^T \mathbf{A}(\mathbf{f} - \hat{\mathbf{f}})\right) d\mathbf{f} \tag{4.20}$$

Here $\hat{\mathbf{f}}$ and $\mathbf{A}$ can be calculated based on equation (4.17) and (4.16).

To calculate the derivatives of the marginal likelihood with respect to $\boldsymbol{\theta}$, we first take the logarithm of both sides of equation (4.20), which gives us,

$$\begin{aligned}
\log q(\mathbf{y}\,|\,\boldsymbol{\theta}) &= \log\left(\Psi\left(\hat{\mathbf{f}}\right)\right) + \log\int \exp\left(-\frac{1}{2}\left(\mathbf{f}-\hat{\mathbf{f}}\right)^T \mathbf{A}\left(\mathbf{f}-\hat{\mathbf{f}}\right)\right)d\mathbf{f} \\
&= \log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right) + \log p\left(\hat{\mathbf{f}}\right) \quad + \quad \frac{N}{2}\log 2\pi - \frac{1}{2}\log|\mathbf{A}| \\
&= \log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right) - \frac{1}{2}\hat{\mathbf{f}}^T\mathbf{K}_{ff}^{-1}\hat{\mathbf{f}} - \frac{1}{2}\log\left|\mathbf{K}_{ff}\right| - \frac{N}{2}\log 2\pi + \frac{N}{2}\log 2\pi - \frac{1}{2}\log|\mathbf{A}| \\
&= \log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right) - \frac{1}{2}\hat{\mathbf{f}}^T\mathbf{K}_{ff}^{-1}\hat{\mathbf{f}} - \frac{1}{2}\log\left|\mathbf{K}_{ff}\right| - \frac{1}{2}\log|\mathbf{A}| \\
&= \log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right) - \frac{1}{2}\hat{\mathbf{f}}^T\mathbf{K}_{ff}^{-1}\hat{\mathbf{f}} - \frac{1}{2}\log\left|\mathbf{K}_{ff}\right|\left|\mathbf{K}_{ff}^{-1} - \nabla\nabla\log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right)\right| \\
&= \log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right) - \frac{1}{2}\hat{\mathbf{f}}^T\mathbf{K}_{ff}^{-1}\hat{\mathbf{f}} - \frac{1}{2}\log|\mathbf{B}|
\end{aligned} \tag{4.21}$$

Here we have explicitly use the representation of $\Psi(\mathbf{f})$ in equation (4.14) and the representation of $\mathbf{A}$ in equation (4.16). For the convenience of discussion we also define a matrix $\mathbf{B}$ as,

$$\mathbf{B} = \mathbf{I}_N + \left(-\nabla\nabla\log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right)\right)^{\frac{1}{2}}\mathbf{K}_{ff}\left(-\nabla\nabla\log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right)\right)^{\frac{1}{2}} \tag{4.22}$$

And so,

$$\begin{aligned}
|\mathbf{B}| &= \left|\mathbf{I}_N + \left(-\nabla\nabla\log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right)\right)^{\frac{1}{2}}\mathbf{K}_{ff}\left(-\nabla\nabla\log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right)\right)^{\frac{1}{2}}\right| \\
&= \left|\mathbf{K}_{ff}\right|\left|\mathbf{K}_{ff}^{-1} - \nabla\nabla\log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right)\right|
\end{aligned} \tag{4.23}$$

The last step of above equaiton is based on equation (9) in the appendix.

The derivatives of $\log q(\mathbf{y}\,|\,\boldsymbol{\theta})$ in equation (4.21) with respect to the hyperparameters $\boldsymbol{\theta}$ depend on two parts: First, the kernel matrix $\mathbf{K}_{ff}$ is an explicit function of $\boldsymbol{\theta}$. Beside that, $\hat{\mathbf{f}}$ and thus $\mathbf{B}$ are also implicit functions of $\boldsymbol{\theta}$. So the derivatives of $\log q(\mathbf{y}\,|\,\boldsymbol{\theta})$ will be obtained by chain rules. After combining the derivatives from the two parts, we obtain,

$$\begin{aligned}\frac{\partial \log q(\mathbf{y}\,|\,\boldsymbol{\theta})}{\partial \theta_j} &= \frac{1}{2}\hat{\mathbf{f}}^T \mathbf{K}_{ff}^{-1} \frac{\partial \mathbf{K}_{ff}}{\partial \theta_j} \mathbf{K}_{ff}^{-1}\hat{\mathbf{f}} - \frac{1}{2} tr\left(\left(\mathbf{K}_{ff} - \left[\nabla\nabla \log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right)\right]^{-1}\right)\frac{\partial \mathbf{K}_{ff}}{\partial \theta_j}\right) \\ &- \sum_{i=1}^{N} \frac{1}{2} \mathbf{A}_{ii}^{-1} \frac{\partial^3}{\partial f_i^3} \log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right)\left[\mathbf{I}_N - \mathbf{K}_{ff}\nabla\nabla \log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right)\right]^{-1} \frac{\partial \mathbf{K}_{ff}}{\partial \theta_j} \nabla \log p\left(\mathbf{y}\,|\,\hat{\mathbf{f}}\right)\end{aligned} \quad (4.24)$$

Here we leave out the calculation details and only give the final result. For more detail please refer to Section 5.5.1 in [Rasmussen and Williams 2006]. From equation (4.24) we can see that besides $\nabla \log p(\mathbf{y}\,|\,\mathbf{f})$ and $\nabla\nabla \log p(\mathbf{y}\,|\,\mathbf{f})$ given in equation (4.18), another key calculation in optimizing the hyperparameters by maximizing $\log q(\mathbf{y}\,|\,\boldsymbol{\theta})$ will be the derivatives of the kernel matrix w.r.t. $\boldsymbol{\theta}$, i.e., $\frac{\partial \mathbf{K}_{ff}}{\partial \boldsymbol{\theta}}$. We will deduct these derivatives for a special kind of kernels when we discuss the SPGPC model in Section 5.1.1.

4.1.5 Difficulties of GP Models

Based on what we discussed so far, the biggest difficulty of GP models are their costly computations at both the training and evaluation stages. Given *N* training instances, the time complexities of the two stages for regressions are $O\left(N^3\right)$ and $O\left(N^2\right)$ respectively, which arise from the inversion of the kernel matrix [Rasmussen and Williams 2006]. In a classification problem, the overhead could be even higher because it takes extra computations to approximate the non-Gaussian distributions. This high computation cost limits the usage of the full GP model to the applications with only small or moderate sizes.

This computation difficulty has been approached in different ways over the past two decades. Especially, a lot of the efforts have been taken to find the effective approximations to the full GP model. In these approximation models, the computation complexity usually scales down to a lower level whereas the accuracy loss is insignificant. For example, the *Sparse Pseudo-*

input Gaussian Process (SPGP) model [Snelson and Ghahramani 2006; Snelson 2007], which serves as the basis of our extension works in the next chapter, has reduced the training and evaluation complexity to $O\left(NM^2\right)$ and $O\left(M^2\right)$ respectively. Here M is the number of the *inducing input vectors*—a set of pseudo vectors in the input space, selected as the *prototypes* of the original input vectors. Generally M is much smaller than the number of training vectors N, and the computation cost is reduced accordingly. A problem with SPGP model is that it is proposed to solve only regression problems. Its extension to classification cases is not straightforward because of the involved approximations. We will introduce SPGP and discuss its extensions in more detail in Chapter 5.

4.2 Sparse Gaussian Process Models

The traditional GP model reviewed above is usually referred to as the *full GP* model in the sense that it uses the *whole* training dataset as its parameters for the later evaluation on new data. As a powerful nonparametric model, full GP is popular in many different applications because it is flexible, simple to implement and can generate fully probabilistic results. However, as we discussed in Section 4.1, the major difficulty with full GP is its high computation costs at the training and the evaluation stage, which are $\mathrm{O}\left(N^3\right)$ and $\mathrm{O}\left(N^2\right)$ respectively [Rasmussen and Williams 2006]. These computation complexities are caused by the inversion of a $N \times N$ matrix $\mathbf{K}_{ff} + \sigma^2\mathbf{I}$ in the model calculation. As a consequence, the full GP model is usually prohibitive on large scale problems that involve hundreds or thousands training instances.

Fortunately, a lot of work has been done over the past two decades to overcome this computation difficulty. Specially, different *sparse GP* models have been proposed, as approximations to the full GP model [Smola and Bartlett 2001; Csato and Opper 2002;

Lawrence, Seeger et al. 2003; Candela and Rasmussen 2005; Keerthi and Chu 2006; Snelson and Ghahramani 2006; Snelson 2007]. It is common to all these sparse models that instead of treating the whole training data exactly, only a subset of them are explicitly used in making new predictions, and the remaining observations are given approximate but cheaper computation treatment. With these approximations, we are able to make the computation costs scale down to a lower level without a significant loss of learning accuracy. In fact, great improvements have been achieved by using these sparse models for regression problems. However, the published sparse models usually have very different motivations and emphasis, which makes their training (inference) algorithms very different from the one used in the traditional full GP. These differences pose the biggest challenge to extend the sparse models into classification problems, as most of the GP related classification techniques such as Laplace/EP framework are strictly based on the inference algorithm of the traditional GP.

Recently Candela and Rasmussen [Candela and Rasmussen 2005] proposed a unifying view of the different sparse GP models. From this viewpoint, the differences among various sparse models can be equivalently viewed as different assumptions about the *joint prior* $p(\mathbf{f}, \mathbf{f}_*)$ in the GP model. Moreover, besides these different assumptions, the unifying framework views the inference algorithms in different sparse models as exactly the same as the one in a full GP. By unifying the sparse models, this framework provides us a means of incorporating the classification techniques originally for the full GP into the different sparse models, although the involved deductions are not trivial.

In the next section we will briefly review this powerful unifying framework. In Chapter 5 we will develop the detailed computations to extend a particular sparse model (i.e., SPGP model) to handle classification problems.

4.3 A Unifying View of Sparse GPs

4.3.1 The General Framework

Assuming the training cases are denoted by $\mathbf{X}$ and $\mathbf{f}$ and the test cases are $\mathbf{X}_*$ and $\mathbf{f}_*$, a full GP model [Williams and Rasmussen 1996; Rasmussen and Williams 2006] essentially gives an assumption about the *joint prior probability* of $\mathbf{f}$ and $\mathbf{f}_*$, i.e.,

$$p(\mathbf{f},\mathbf{f}_*) = \mathrm{N}\left(0,\begin{bmatrix}\mathbf{K}_{ff} & \mathbf{K}_{f*}\\ \mathbf{K}_{*f} & \mathbf{K}_{**}\end{bmatrix}\right) \tag{4.25}$$

Here the different parts of the covariance matrix are calculated by evaluating a specific *kernel function* $k(\mathbf{x},\mathbf{x}')$ on the corresponding input data, e.g., $\mathbf{K}_{ff} = k(\mathbf{X},\mathbf{X})$, $\mathbf{K}_{f*} = k(\mathbf{X},\mathbf{X}_*)$, $\mathbf{K}_{**} = k(\mathbf{X}_*,\mathbf{X}_*)$, and $\mathbf{K}_{f*} = \mathbf{K}_{*f}^T$. And we will see in the following discussions that under the unifying framework the different sparse GPs essentially correspond to different assumptions of $p(\mathbf{f},\mathbf{f}_*)$.

With the joint prior assumption in Equation (4.25), the prediction task for regression problems corresponds to calculating the posterior probability $p(\mathbf{f}_*\,|\,\mathbf{y})$, where $\mathbf{y}$ are the observed output values. The output values for regressions are usually modeled as an underlying outputs interrupted by an additive Gaussian noise, i.e.,

$$p(\mathbf{y}\,|\,\mathbf{f}) = N(\mathbf{f},\sigma^2\mathbf{I}) \tag{4.26}$$

Here variance σ^2 represents the noise level. To calculate $p(\mathbf{f}_*\,|\,\mathbf{y})$, we first combine equations (4.25) and (4.26) to obtain the joint probability of $\mathbf{y}$ and $\mathbf{f}_*$, which is given by,

$$p(\mathbf{y},\mathbf{f}_*) = \mathrm{N}\left(0, \begin{bmatrix} \mathbf{K}_{ff}+\sigma^2\mathbf{I} & \mathbf{K}_{f*} \\ \mathbf{K}_{*f} & \mathbf{K}_{**} \end{bmatrix}\right) \tag{4.27}$$

After that, finding $p(\mathbf{f}_* \mid \mathbf{y})$ simply corresponds to deriving the conditional Gaussian distribution from the joint. Based on equation (3) in the appendix, $p(\mathbf{f}_* \mid \mathbf{y})$ can be represented in close-form as

$$p(\mathbf{f}_* \mid \mathbf{y}) = \mathrm{N}\left(\mathbf{K}_{*f}\left(\mathbf{K}_{ff}+\sigma^2\mathbf{I}\right)^{-1}\mathbf{y}, \mathbf{K}_{**}-\mathbf{K}_{*f}\left(\mathbf{K}_{ff}+\sigma^2\mathbf{I}\right)^{-1}\mathbf{K}_{f*}\right) \tag{4.28}$$

From equation (4.28) we can see that the major computation cost of the full GP lies in the inversion of the $N \times N$ matrix $\left(\mathbf{K}_{ff}+\sigma^2\mathbf{I}\right)$, which usually requires $\mathrm{O}\left(N^3\right)$ operations. Here N is the training sample size. A full GP model is also called *non-degenerate* GP [Candela and Rasmussen 2005] in the literature, as the training kernel matrix $\mathbf{K}_{ff}$ is usually of *full rank*. To reduce this computation cost incurred by inverting the corresponding matrix, we can use a lower-rank approximation to $\mathbf{K}_{ff}$, e.g.,

$$\mathbf{K}_{ff} = \mathbf{Q}\mathbf{Q}^T \tag{4.29}$$

where $\mathbf{Q}$ is an $N \times M$ matrix with a maximum rank $M < N$. Equation (4.29) is a rational approximation in practice because the underlying input-output function is usually smooth, which implies that the eigenvalue spectrum of the full rank matrix $\mathbf{K}_{ff}$ usually has a rapid decay.

Based on equation (4.29), the inversion $\left(\mathbf{K}_{ff}+\sigma^2\mathbf{I}\right)^{-1}$ can be recast by using equation (8) in the appendix, which is given by

$$\begin{aligned}\left(\mathbf{K}_{ff}+\sigma^2\mathbf{I}_N\right)^{-1} &\approx \left(\mathbf{Q}\mathbf{Q}^T+\sigma^2\mathbf{I}_N\right)^{-1} \\ &= \sigma^{-2}\mathbf{I}_N-\sigma^{-2}\mathbf{Q}\left(\sigma^2\mathbf{I}_M+\mathbf{Q}^T\mathbf{Q}\right)^{-1}\mathbf{Q}^T\end{aligned} \tag{4.30}$$

Here $\mathbf{I}_N$ and $\mathbf{I}_M$ are the $N\times N$ and $M\times M$ identity matrix respectively. From equation (4.30), we can see that the original $N\times N$ matrix inversion problem has been converted to the inversion of a $M\times M$ matrix $\left(\sigma^2\mathbf{I}_M+\mathbf{Q}^T\mathbf{Q}\right)^{-1}$. Accordingly, the computation complexity has been reduced from $\mathrm{O}\left(N^3\right)$ to $\mathrm{O}\left(NM^2\right)$.

The reduced-rank approximation in equation (4.29) itself can be obtained in several ways. For example, by using the *Frobenius norm* [Golub and Loan 1989], $\mathbf{K}_{ff}$ can be approximated in the form $\mathbf{U}_M\mathbf{\Lambda}_M\mathbf{U}_M^T$, where $\mathbf{\Lambda}_M$ is the diagonal matrix of the leading M eigenvalues of $\mathbf{K}_{ff}$ and $\mathbf{U}_M$ is the matrix of the corresponding orthonormal eigenvectors. However, this approximation is usually of limited interest in practice because the eigen-decomposition of $\mathbf{K}_{ff}$ itself needs $\mathrm{O}\left(N^3\right)$ operations.

The recently proposed unifying framework [Candela and Rasmussen 2005] and the sparse GP models offer another way of doing this kind of approximation. A lot of existing sparse GP models can be viewed as special cases of this general framework. Its underlying assumption is that the dependencies among all data in GP assumption can be *induced* by only a finite number of hidden variables. In a sparse GP model, these hidden variables are usually a set of pseudo input vectors $\mathbf{X}_u$ called *inducing vectors*. And the corresponding outputs of $\mathbf{X}_u$ are called *latent variables*, which is denoted by vector $\mathbf{u}$. In general, $\mathbf{u}$ plays a similar role as the training output values but they are not directly observed in the dataset. As the distribution of $\mathbf{u}$ is essentially determined by the values of $\mathbf{X}_u$, $\mathbf{X}_u$ is usually viewed as the additional hyper-parameters in addition to those hyper-parameters $\boldsymbol{\theta}$ that are introduced by the kernel function $k(\mathbf{x},\mathbf{x}')$. Several approaches have been proposed to determine $\mathbf{X}_u$ and $\boldsymbol{\theta}$. They will be discussed in detail in Section 4.3.6.

To see how the introduction of the latent variables enables us to approximate the training kernel matrix, we rewrite the joint prior in equation (4.25) based on $\mathbf{u}$, that is,

$$p(\mathbf{f},\mathbf{f}_*) = \int p(\mathbf{f},\mathbf{f}_* \mid \mathbf{u})\, p(\mathbf{u})\, d\mathbf{u} \tag{4.31}$$

Equation (4.31) is based on GP's *marginalization* property [Rasmussen and Williams 2006]. In other words, we are essentially assuming that the variables $\mathbf{u}$ are generated from the same Gaussian process that generates $\mathbf{f}$ and $\mathbf{f}_*$. Thus, the prior of $\mathbf{u}$ can be represented by

$$p(\mathbf{u}) = \mathrm{N}\left(\mathbf{0},\mathbf{K}_{uu}\right) \tag{4.32}$$

where $\mathbf{K}_{uu}$ is the covariance matrix on $\mathbf{X}_u$, and is generated by the same kernel function $k(\mathbf{x},\mathbf{x}')$ that generates $\mathbf{K}_{ff}$, $\mathbf{K}_{f*}$ and $\mathbf{K}_{**}$.

So far there haven't been any approximations involved in equation (4.31) and (4.32), all the inferences are still exact. The key step that gives rise to the approximation and thus the sparseness is a further assumption of the conditional independence between $\mathbf{f}$ and $\mathbf{f}_*$, that is,

$$p(\mathbf{f},\mathbf{f}_*) \approx q(\mathbf{f},\mathbf{f}_*) = \int q(\mathbf{f} \mid \mathbf{u})\, q(\mathbf{f}_* \mid \mathbf{u})\, p(\mathbf{u})\, d\mathbf{u} \tag{4.33}$$

Here we denote the real *probability distribution function* (pdf) by $p(\cdot)$ and the approximation pdf by $q(\cdot)$. Comparing equation (4.31) with (4.33) we can see that in the latter equation the conditional joint probability $p(\mathbf{f},\mathbf{f}_* \mid \mathbf{u})$ has been approximated by the product $q(\mathbf{f}\,|\,\mathbf{u})q(\mathbf{f}_*\,|\,\mathbf{u})$. This essentially implies that $\mathbf{f}$ and $\mathbf{f}_*$ are *independent* conditioning on the latent variables $\mathbf{u}$. In other words, the variable $\mathbf{u}$ induces the dependencies between the training and test instances.

Under the unifying framework, equation (4.33) is the key assumption common to all the sparse models. Different sparse models just correspond to different additional assumptions about the specific forms of $q(\mathbf{f}\,|\,\mathbf{u})$ and $q(\mathbf{f}_*\,|\,\mathbf{u})$. These different specifications of $q(\mathbf{f}\,|\,\mathbf{u})$ and $q(\mathbf{f}_*\,|\,\mathbf{u})$ can be viewed as the different approximations to the following *exact forms*,

$$p(\mathbf{f}\,|\,\mathbf{u}) = \mathrm{N}\left(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{u}, \mathbf{K}_{ff} - \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf}\right) \tag{4.34}$$

$$p(\mathbf{f}_*\,|\,\mathbf{u}) = \mathrm{N}\left(\mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\mathbf{u}, \mathbf{K}_{**} - \mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\mathbf{K}_{u*}\right) \tag{4.35}$$

Equations (4.34) and (4.35) are exact and can be derived directly based on the assumption that $\mathbf{u}$, $\mathbf{f}$ and $\mathbf{f}_*$ are generated by the same GP.

In equation (4.33), because the real $p(\mathbf{f},\mathbf{f}_*)$ is approximated by $q(\mathbf{f},\mathbf{f}_*)$, the training kernel matrix $\tilde{\mathbf{K}}$ in $q(\mathbf{f},\mathbf{f}_*)$ depends on the inducing vectors $\mathbf{X}_u$ now. As the number of inducing vectors is usually smaller than the number of training vectors, $\tilde{\mathbf{K}}$ is not of full rank anymore. In this way the full-rank covariance matrix in $p(\mathbf{f},\mathbf{f}_*)$ has been approximated by the lower-rank $\tilde{\mathbf{K}}$, which will hopefully end up with a reduced computation cost.

In the following several sections, we will briefly review some specific types of sparse GP Except the SD model introduced in Section 4.3.2, all the other sparse models can be viewed as special cases of this general framework, which correspond to specific assumptions about $q(\mathbf{f}\,|\,\mathbf{u})$ and $q(\mathbf{f}_*\,|\,\mathbf{u})$.

4.3.2 The Subset of Data (SD) Model

The SD sparse model is different from other sparse models in that it is not a special case of the general framework introduced above. However, because of its easy implementation and special relation to full GP, it is widely used as a benchmark when compared to other more sophisticated sparse models [Candela and Rasmussen 2005].

Intuitively the SD model can be viewed as a shrunk version of full GP, in the sense that the SD model is an exact full GP, but applied only to a subset of training data. For a training set with N observations, the SD model will choose M instances from them and totally ignore the other $N-M$ instances. Since all the inference steps in SD model are exactly the same as the full GP, there are no extra approximation overheads introduced. Compared to those more sophisticated sparse models, this could be a crucial advantage in practice, especially when the model's time complexity is of the most important. The training and evaluation complexity of SD are $\mathrm{O}\left(M^3\right)$ and $\mathrm{O}\left(M^2\right)$ respectively, where M is the number of training instances used by SD.

The main disadvantage of the SD model is also obvious. Compared to other sparse models, the SD model totally discards a part of the training data, so it is expected that its accuracy is generally lower. Recently, there have been several advanced extensions of the basic SD model proposed in the literature. One typical representative is the Informative Vector Machine (IVM) proposed by [Lawrence, Seeger et al. 2003]. Compared to the basic SD model, IVM can significantly improve the learning accuracy but at a cost of a higher computation overhead.

4.3.3 The Subset of Regressors (SR) Model

After it was first proposed by Silverman [Silverman 1985] and again by [Wahba, Lin et al. 1999], the Subset of Regressors (SR) algorithm has been further adapted by [Smola and Bartlett 2001] to as a sparse greedy approximation to the full GP model. The SR models can be viewed as a special case of the unifying framework in Section 4.3.1 with the additional assumptions about the conditional priors $q(\mathbf{f}\,|\,\mathbf{u})$ and $q(\mathbf{f}_*\,|\,\mathbf{u})$ expressed as,

$$q_{SR}(\mathbf{f}\,|\,\mathbf{u}) = \mathrm{N}\left(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{u}, \mathbf{0}\right) \tag{4.36}$$

$$q_{SR}(\mathbf{f}_*\,|\,\mathbf{u}) = \mathrm{N}\left(\mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\mathbf{u}, \mathbf{0}\right) \tag{4.37}$$

We can see the differences by comparing these assumptions with the exact distributions of a full GP in equation (4.34) and (4.35). In particular, the two assumptions here essentially suggest a linear and deterministic relationship between $\mathbf{f}/\mathbf{f}_*$ and $\mathbf{u}$, because the covariance of both conditionals are exactly $\mathbf{0}$.In other words, in addition to the assumption that the dependencies between $\mathbf{f}$ and $\mathbf{f}_*$ is only induced by $\mathbf{u}$, here the SR model further assumes that these dependencies are deterministic.

Substituting these specific assumptions into the general framework in equation (4.33), we obtain the corresponding joint prior assumption of the SR models,

$$\begin{aligned} q_{SR}(\mathbf{f},\mathbf{f}_*) &= \int q_{SR}(\mathbf{f}\,|\,\mathbf{u})\, q_{SR}(\mathbf{f}_*\,|\,\mathbf{u})\, p(\mathbf{u})\, d\mathbf{u} \\ &= \int \mathrm{N}\left(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{u},\mathbf{0}\right) \mathrm{N}\left(\mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\mathbf{u},\mathbf{0}\right) \mathrm{N}\left(\mathbf{0},\mathbf{K}_{uu}\right) d\mathbf{u} \\ &= \mathrm{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{Q}_{ff} & \mathbf{Q}_{f*} \\ \mathbf{Q}_{*f} & \mathbf{Q}_{**} \end{bmatrix}\right) \end{aligned} \tag{4.38}$$

where the corresponding sub-matrix in the covariance are defined as,

$$\mathbf{Q}_{ab} \triangleq \mathbf{K}_{au}\mathbf{K}_{uu}^{-1}\mathbf{K}_{ub} \tag{4.39}$$

And the last step of equation (4.38) is based on the Gaussian product formula in equation (4) of appendix. Equation (4.38) gives the equivalent prior joint assumption of the SR model. With this assumption, the same inferences in full GP will also apply to the SR model. Thus, to get the predictive distribution $q_{SR}\left(\mathbf{f}_* \mid \mathbf{y}\right)$, we just follow the same inference steps used in the derivation of equation (4.28), which finally gives us,

$$q_{SR}\left(\mathbf{f}_* \mid \mathbf{y}\right) = \mathrm{N}\left(\mathbf{Q}_{*f}\left(\mathbf{Q}_{ff} + \sigma^2\mathbf{I}\right)^{-1}\mathbf{y}, \mathbf{Q}_{**} - \mathbf{Q}_{*f}\left(\mathbf{Q}_{ff} + \sigma^2\mathbf{I}\right)^{-1}\mathbf{Q}_{f*}\right) \tag{4.40}$$

Comparing $q_{SR}\left(\mathbf{f}_* \mid \mathbf{y}\right)$ with the corresponding distribution $p\left(\mathbf{f}_* \mid \mathbf{y}\right)$ of a full GP (in equation (4.28)), we can see that in SR, the trouble-making matrix inversion operation $\left(\mathbf{K}_{ff} + \sigma^2\mathbf{I}\right)^{-1}$ in equation (4.28) has been converted to another matrix inversion $\left(\mathbf{Q}_{ff} + \sigma^2\mathbf{I}\right)^{-1}$. Assume the number of inducing vectors $M < N$. Based on the definition of $\mathbf{Q}_{ff}$ given in equation (4.39), $\mathbf{Q}_{ff}$ is at most of rank M because $\mathbf{K}_{fu}$, $\mathbf{K}_{uu}$ and $\mathbf{K}_{uf}$ are of size $N \times M$, $M \times M$ and $M \times N$ respectively. Thus, the corresponding computation cost of the matrix inversion has been reduced from $\mathrm{O}\left(N^3\right)$ to $\mathrm{O}\left(M^2N\right)$.

Compared to SD, the SR model usually has a better learning accuracy. Unlike SD that simply discards a part of the training instances, the SR model assumes a deterministic linear relationship between the inducing vectors $\mathbf{X}_u$ and the other training instances, and thus is able to capture more information through the averaging effect of the linear relationship.

However, there is an undesired property in SR's result—when a test input $\mathbf{x}_*$ gets further away from the inducing vectors $\mathbf{X}_u$, the variance of the predictive distribution for $\mathbf{x}_*$ in equation (4.40) becomes smaller. To see how this happens, we take a further step to convert the covariance representation in equation (4.40) to,

$$\begin{aligned}&\mathbf{Q}_{**}-\mathbf{Q}_{*f}\left(\mathbf{Q}_{ff}+\sigma^2\mathbf{I}\right)^{-1}\mathbf{Q}_{f*}\\&=\mathbf{K}_{*u}\left(\sigma^{-2}\mathbf{K}_{uf}\mathbf{K}_{fu}+\mathbf{K}_{uu}\right)^{-1}\mathbf{K}_{u*}\end{aligned} \tag{4.41}$$

Here $\left(\sigma^{-2}\mathbf{K}_{uf}\mathbf{K}_{fu}+\mathbf{K}_{uu}\right)^{-1}$ is independent of the test inputs $\mathbf{X}_*$, and $\mathbf{K}_{*u}=k\left(\mathbf{X}_*,\mathbf{X}_u\right)$, $\mathbf{K}_{u*}=k\left(\mathbf{X}_u,\mathbf{X}_*\right)$. Since a kernel function $k\left(\mathbf{x},\mathbf{x}'\right)$ is supposed to be smaller when the two inputs $\mathbf{x}$ and $\mathbf{x}'$ get further, the variance in equation (4.41) will also get smaller when $\mathbf{X}_*$ is distributed further away from $\mathbf{X}_u$.

This result is anti-intuitive to our commonsense knowledge, as what we actually expect is that when a test input is in a region further from the inducing vectors, the confidence of the linear interpolation of equation (4.37) should get weaker, which in turn results in an increase (instead of a decrease) of the prediction uncertainty (measured by the variance/covariance). This unpleasant property of SR can be viewed as a result of the *degenerate* approximation in equation (4.38). Compared to the full GP joint prior in equation (4.25), the approximation $q_{SR}\left(\mathbf{f},\mathbf{f}_*\right)$ has a degenerate covariance matrix as it is not of full rank anymore. Accordingly the Gaussian process behind $q_{SR}\left(\mathbf{f},\mathbf{f}_*\right)$ is a degenerate GP in that its equivalent kernel function has only a finite number of non-zero eigenvalues. Generally speaking, this kind of anti-intuitive result is with all degenerate GP models.

4.3.4 The Projected Process (PP) Model

Just like other sparse models, the PP model has been repeatedly and independently invented several times by different people under different names [Csato and Opper 2002; Seeger, Williams et al. 2003; Rasmussen and Williams 2006]. Here we follow its naming convention introduced in [Rasmussen and Williams 2006]. Under the unifying framework, the PP model essentially corresponds to the following specific assumptions,

$$q_{PP}\left(\mathbf{f}\,|\,\mathbf{u}\right) = \mathrm{N}\left(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{u}, \mathbf{0}\right) \tag{4.42}$$

$$q_{PP}\left(\mathbf{f}_*\,|\,\mathbf{u}\right) = \mathrm{N}\left(\mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\mathbf{u}, \mathbf{K}_{**} - \mathbf{Q}_{**}\right) \tag{4.43}$$

where $\mathbf{Q}_{**}$ is defined in equation (4.39). From these assumptions, we can see that the PP model is very similar to the SR model except that their conditionals of the test output $q_{SR}\left(\mathbf{f}_*\,|\,\mathbf{u}\right)$ and $q_{PP}\left(\mathbf{f}_*\,|\,\mathbf{u}\right)$ are different. Instead of assuming a deterministic relationship as in $q_{SR}\left(\mathbf{f}_*\,|\,\mathbf{u}\right)$, here $q_{PP}\left(\mathbf{f}_*\,|\,\mathbf{u}\right)$ has a non-zero covariance $\mathbf{K}_{**} - \mathbf{Q}_{**}$, which is of full rank due to the presence of the full-rank matrix $\mathbf{K}_{**}$ in it. In fact, the test conditional $q_{PP}\left(\mathbf{f}_*\,|\,\mathbf{u}\right)$ here is exactly the same as the test conditional of a full GP in equation (4.35).

By substituting these specific assumptions into (4.33), we get the equivalent joint prior of the PP model, which is,

$$\begin{aligned} q_{PP}\left(\mathbf{f},\mathbf{f}_*\right) &= \int q_{PP}\left(\mathbf{f}\,|\,\mathbf{u}\right) q_{PP}\left(\mathbf{f}_*\,|\,\mathbf{u}\right) p\left(\mathbf{u}\right) d\mathbf{u} \\ &= \int \mathrm{N}\left(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{u}, \mathbf{0}\right) \mathrm{N}\left(\mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\mathbf{u}, \mathbf{K}_{**} - \mathbf{Q}_{**}\right) \mathrm{N}\left(\mathbf{0}, \mathbf{K}_{uu}\right) d\mathbf{u} \\ &= \mathrm{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{Q}_{ff} & \mathbf{Q}_{f*} \\ \mathbf{Q}_{*f} & \mathbf{K}_{**} \end{bmatrix}\right) \end{aligned} \tag{4.44}$$

Accordingly, the predictive distribution for $\mathbf{f}_*$ in the PP model can be derived as,

$$q_{PP}\left(\mathbf{f}_*\,|\,\mathbf{y}\right) = \mathrm{N}\left(\mathbf{Q}_{*f}\left(\mathbf{Q}_{ff} + \sigma^2\mathbf{I}\right)^{-1}\mathbf{y}, \mathbf{K}_{**} - \mathbf{Q}_{*f}\left(\mathbf{Q}_{ff} + \sigma^2\mathbf{I}\right)^{-1}\mathbf{Q}_{f*}\right) \tag{4.45}$$

We can see that $q_{SR}\left(\mathbf{f}_* \mid \mathbf{y}\right)$ and $q_{PP}\left(\mathbf{f}_* \mid \mathbf{y}\right)$ have the same formulation of the predictive mean. However, due to the existence of the *non-degenerate* $\mathbf{K}_{**}$ in $q_{PP}\left(\mathbf{f},\mathbf{f}_*\right)$, the predictive covariance in $q_{PP}\left(\mathbf{f}_* \mid \mathbf{y}\right)$ is now of full rank. As a result, the anti-intuitive property in SR's result is not present here anymore. A further analysis reveals that the predictive variance in PP is actually always larger than the one in SR. To see why, notice that compared to the predictive distribution in equation (4.40), here the difference in the variance terms is $\mathbf{K}_{**}-\mathbf{Q}_{**}$. And it turns out that $\mathbf{K}_{**}-\mathbf{Q}_{**}$ is always positive definite because it is actually the variance of the conditional probability of $p\left(\mathbf{f}_* \mid \mathbf{u}\right)$ in equation (4.35).

By using a different conditional assumption for the test data, the PP model removes the undesired anti-intuitive property in SR's result. But it also incurs another issue. Since the representation forms for the training and test data in equation (4.42) and (4.43) are different, the underlying stochastic process behind PP is actually not an exact Gaussian process anymore. By definition, a Gaussian process must have a unified representation form for arbitrary data [Candela and Rasmussen 2005]. As a consequence, the PP model could show weird behaviors as the underlying learning model now depends on whether the data is from training or testing set.

Based on equation (4.45), the computation cost of the PP model is also $\mathrm{O}\left(M^2N\right)$. Compared to the full GP a reduction has been made given the number of inducing vectors M is usually smaller than the total number of training cases N .With some pre-computations, the calculations of the predictive mean and variance can be further reduced to $\mathrm{O}\left(M\right)$ and $\mathrm{O}\left(M^2\right)$ respectively.

4.3.5 The Sparse Pseudo-input Gaussian Process (SPGP) Model

Recently [Snelson and Ghahramani 2006; Snelson 2007] proposes another sparse GP model called the Sparse Pseudo-input Gaussian Process (SPGP), which is able to eliminate the anti-intuitive property in SR's result and also abide by the consistency of GP's definition that has been violated by PP. In details, the SPGP model corresponds to the following joint prior assumptions,

$$q_{SPGP}\left(\mathbf{f}\,|\,\mathbf{u}\right) = \mathrm{N}\left(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{u}, \mathrm{diag}\left[\mathbf{K}_{ff} - \mathbf{Q}_{ff}\right]\right) \tag{4.46}$$

$$q_{SPGP}\left(\mathbf{f}_*\,|\,\mathbf{u}\right) = \mathrm{N}\left(\mathbf{K}_{*u}\mathbf{K}_{uu}^{-1}\mathbf{u}, \mathrm{diag}\left[\mathbf{K}_{**} - \mathbf{Q}_{**}\right]\right) \tag{4.47}$$

Here the matrix operator $\mathrm{diag}\left[\mathbf{A}\right]$ is the matrix that keeps only the diagonal part of $\mathbf{A}$ and leaves all the non-diagonal entries as exactly 0. Based on these assumptions, we can derive the equivalent joint prior of SPGP as,

$$\begin{aligned} & q_{SPGP}\left(\mathbf{f},\mathbf{f}_*\right) \\ & = \int q_{SPGP}\left(\mathbf{f}\,|\,\mathbf{u}\right) q_{SPGP}\left(\mathbf{f}_*\,|\,\mathbf{u}\right) p\left(\mathbf{u}\right) d\mathbf{u} \\ & = \mathrm{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{Q}_{ff} + \mathrm{diag}\left[\mathbf{K}_{ff} - \mathbf{Q}_{ff}\right] & \mathbf{Q}_{f*} \\ \mathbf{Q}_{*f} & \mathbf{Q}_{**} + \mathrm{diag}\left[\mathbf{K}_{**} - \mathbf{Q}_{**}\right] \end{bmatrix}\right) \end{aligned} \tag{4.48}$$

It is interesting to compare equation (4.48) with the corresponding joint priors of other GP models. In fact, given the full GP's covariance matrix $\mathbf{K}_{full-GP} = \begin{bmatrix} \mathbf{K}_{ff} & \mathbf{K}_{f*} \\ \mathbf{K}_{*f} & \mathbf{K}_{**} \end{bmatrix}$ and the SR model's covariance $\mathbf{K}_{SR} = \begin{bmatrix} \mathbf{Q}_{ff} & \mathbf{Q}_{f*} \\ \mathbf{Q}_{*f} & \mathbf{Q}_{**} \end{bmatrix}$, the covariance in $q_{SPGP}\left(\mathbf{f},\mathbf{f}_*\right)$ can be viewed as an combination of the two models in the sense that its diagonal part and non-diagonal part are from $\mathbf{K}_{full-GP}$ and $\mathbf{K}_{SR}$ respectively. With these assumptions, the predictive distribution of SPGP now becomes,

$$q_{SPGP}\left(\mathbf{f}_* \mid \mathbf{y}\right) = \mathrm{N}\left(\mathbf{Q}_{*f}\left(\mathbf{Q}_{ff}+\mathbf{\Lambda}\right)^{-1}\mathbf{y}, \mathbf{K}_{**} - \mathbf{Q}_{*f}\left(\mathbf{Q}_{ff}+\mathbf{\Lambda}\right)^{-1}\mathbf{Q}_{f*}\right) \tag{4.49}$$

where $\mathbf{\Lambda}$ is the diagonal matrix given by,

$$\mathbf{\Lambda} = \mathrm{diag}\left[\mathbf{K}_{ff} - \mathbf{Q}_{ff} + \sigma^2\mathbf{I}\right] \tag{4.50}$$

The covariance of $q_{SPGP}\left(\mathbf{f}_* \mid \mathbf{y}\right)$ is also of full rank because of its diagonal part in matrix $\mathbf{K}_{**}$, thus the SPGP model is not degenerate. And also because the joint prior in equation (4.48) has a consistent representation for both training and testing data, the SPGP is a consistent GP. At the first glance, the computation reduction in the inversion $\left(\mathbf{Q}_{ff}+\mathbf{\Lambda}\right)^{-1}$ is not obvious as $\mathbf{\Lambda}$ itself is now of full rank. But it turns out that with some calculation tricks this computation cost can be reduced to exactly $\mathrm{O}\left(M^2N\right)$, just as the same as in other sparse models. Here the key point is that the inversion of $\mathbf{\Lambda}$ can be done in $\mathrm{O}\left(N\right)$ operations because it is a diagonal matrix. More details can be found in [Snelson 2007].

4.3.6 Determinations of Inducing Variables

Under the general inference framework discussed above, the SR, PP, and SPGP models can all be viewed as special assumptions based on the conditionally impendent relationship in equation (4.33). Considering this relationship is a relatively strong assumption, it is reasonable to expect that the locations of the inducing variables $\mathbf{X}_u$ will have a significant influence on those sparse models' performances. In fact, there have been a lot of algorithms and heuristics proposed in the literature, aiming at determining these inducing variables in a systematic way. For example, in the SD, SR and PP models, the inducing vectors $\mathbf{X}_u$ are usually selected as a subset of the training cases, which is also exemplified in another popular nonparametric model—the *Support Vector Machine* (SVM) [Vapnik 1995; Scholkopf and Smola 2002]. As the combinatorial optimizations involved in this process could be prohibitive, a greedy approach is usually exploited on various selection criteria such as *online learning* [Csato and Opper 2002], *greedy posterior maximization* [Smola and Bartlett 2001], *maximum information gain* [Seeger, Williams et al. 2003], *matching pursuit* [Keerthi and Chu 2006] and etc.

On the other hand, a specific inducing-vector determination algorithm has been proposed along with the SPGP model [Snelson and Ghahramani 2006]. In details, the inducing vectors $\mathbf{X}_u$ are treated the same way as the other hyper-parameters $\boldsymbol{\theta}$ that are introduced by the normal kernel function. These two types of the parameters are optimized together at the model selection stage, through the maximization of the *marginal likelihood* given by,

$$p(\mathbf{y} \mid \boldsymbol{\theta}) = \int p(\mathbf{y} \mid \mathbf{f})\, p(\mathbf{f})\, d\mathbf{f} \tag{4.51}$$

Here $\mathbf{y}$ are the observed training outputs and $\mathbf{f}$ are their underlying values modeled by Gaussian process. In a regression case, $\mathbf{y}$ and $\mathbf{f}$ are related by the assumption that $p(\mathbf{y} \mid \mathbf{f}) = \mathrm{N}\left(\mathbf{f}, \sigma^2 \mathbf{I}\right)$. In a sparse GP model such as SPGP, the equivalent GP prior that $\mathbf{f}$ follows is parameterized by both $\mathbf{X}_u$ and $\boldsymbol{\theta}$. Here the inducing inputs $\mathbf{X}_u$ are not necessarily a subset of the training instances anymore. Equation (4.51) can be used to serve an optimization purpose to estimate these parameters, as $p(\mathbf{y} \mid \boldsymbol{\theta})$ is independent of the test data. More details on optimizing the marginal likelihood for this kind of purpose can be found in [MacKay 2003; Bishop 2006; Rasmussen and Williams 2006].

By relaxing the constraint that the inducing variables must be a subset of training instances, the above method in SPGP converts a discrete combinatorial problem into a continuous optimization. As a result, the SPGP model has the potential to give better performances than the others since its search space is essentially larger. Besides, by optimizing the two types of parameters together, it can avoid the unpleasant convergence problem that happens when the optimizations of different types of parameters are interleaving. On the downside, maximizing equation (4.51) needs the evaluation of the corresponding derivatives of the SPGP's covariance, which is an $\mathrm{O}\left(M^2N+MND\right)$ time operation in general on a dataset with N training instances, M inducing vectors and D input features. The extra computation overhead not only increases the model's training time but also increases the model's risk of suffering the curse of dimensionality. Thus, compared to other sparse models, an appropriate feature selection step is usually crucial to the SPGP model.

4.4 Summary

In this section we discuss a specific nonparametric model—the Gaussian Process model. As a full Bayesian nonparametric model, GP has several advantages over the traditional parametric models, such as the flexibility that can grow with training data, the expressive prediction results and the special way of tuning hyperparameters.

Table 4-1: List of popular extensions of GP

Original Model	*Extended Models*	*Classification Model?*	*Reduced Computation Cost?*
GP (nonparametric)	Laplace Approximation [Rasmussen and Williams 2006]	√	
	Expectation Propagation [Minka 2001]	√	
	Sparse Greedy GP Regression [Smola and Bartlett 2001]		√
	Projected Process Model [Keerthi and Chu 2006]		√

	SPGP Regression Model [Snelson and Ghahramani 2006]		√
	SPGP Classifier (Chapter 5)	√	√

However, GP models also have some practical difficulties with large-scale data sets, especially when the number of input features is also high. Here the main difficulty of the full GP model is its high computation costs involved in the training and evaluation stages when there are a large number of instances. For regression problems, different sparse GP models have been proposed aiming at reducing this kind of computation cost. Some of them are reviewed in this chapter and listed in Table 4-1. However, the further extensions of these sparse models for classifications are still challenging, due to the extra computation complexity involved in GP's approximation frameworks for classifications. Thus, inspired by the SPGP regression model, we proposed the SPGP classifier (SPGPC) to handle the high computation cost issue for classification problems. In the next chapter, we will introduce in detail the proposed SPGPC model.

Chapter 5

5 Modification of the SPGP Regression Model for Classification Problems

In the last chapter we reviewed several sparse GP models for regression problems, in the context of the unifying framework proposed by [Candela and Rasmussen 2005]. In this chapter, we take a further step to extend a particular sparse model, namely the SPGP model [Snelson and Ghahramani 2006], to handle classification problems. Besides, by using the specific *automatic relevance determination* (ARD) kernel, we can provide useful model selection results with the proposed model, and thus further reduce SPGP's vulnerability to the curse of dimensionality. Experimental results from both the benchmark and real-world applications show the potentials of the proposed SPGP Classifier (SPGPC) model.

5.1 Extension of SPGP for Classification

Thanks to the framework proposed by [Candela and Rasmussen 2005], the different sparse GP models have been unified into the same inference scheme with different assumptions of the prior probabilities. This actually provides the possibility of extending these sparse models to classifications, as all the existing GP classification frameworks such as the *Expectation Propagation* (EP) [Minka 2001] and the *Laplace Approximation* [Rasmussen and Williams 2006] are based on this inference scheme. However, just as we proposed ENRBFC in Chapter 3, some details of this extension work still need to be handled carefully due to the difference between regressions and classifications that are discussed in Section 1.3. For example, the approximation steps involved in the Laplace approximation need to compute the derivatives of covariance matrices with respect to model's hyperparameters. The development of these computations usually poses mathematical challenges in practice.

In this section, we develop the requisite computations to propose the SPGP Classifier (SPGPC) model. The SPGPC model is based on the SPGP regression model that is discussed in the last chapter. Specially, we compute in detail the quantities that are needed to embed SPGP model to the Laplace approximation framework that is discussed in Section 4.1.3 and 4.1.4. In principle the developments here can also be used to extend other sparse regression models, but we are especially interested in the SPGP model because of some pleasant characteristics it has (more details can be found in Section 4.3.5 and 4.3.6). Furthermore, the Automatic Relevance Determination (ARD) kernel [Neal 1996] is used with the SPGPC model to generate useful feature selection results. As discussed in Section 4.3.6 these feature selection results could be crucial to the SPGP and SPGPC model, as their computations depend on both the number of instances and the number of input features. We also provide some experiment analysis in Section 5.3 to compare SPGPC with other sparse GP based classification models.

5.1.1 SPGP Model for Classification

To extend SPGP with the existing classification frameworks such as the Laplace approximation, there are two kinds of computations that are needed, namely the evaluation of the kernel matrix itself and the evaluation of its derivatives wrt the kernel hyper-parameters $\boldsymbol{\theta}$ and inducing vectors $\mathbf{X}_u$. As the first evaluation of the kernel itself is relatively simple and straightforward, we fill focus on the second computation in this section. Please note these calculations of the derivatives wrt $\boldsymbol{\theta}$ and $\mathbf{X}_u$ will also be used when optimizing them by maximizing the marginal likelihood.

The assumption of the joint prior of SPGP is represented by equation (4.48) in Section 4.3.5, which is equivalent to using the following kernel function in a full GP,

$$k_{SPGP}\left(\mathbf{x},\mathbf{x}' \mid \mathbf{X}_u,\boldsymbol{\theta}\right) = Q\left(\mathbf{x},\mathbf{x}' \mid \mathbf{X}_u,\boldsymbol{\theta}\right) + \delta\left(\mathbf{x},\mathbf{x}'\right)\left[k\left(\mathbf{x},\mathbf{x}' \mid \boldsymbol{\theta}\right) - Q\left(\mathbf{x},\mathbf{x}' \mid \mathbf{X}_u,\boldsymbol{\theta}\right)\right] \quad (5.1)$$

where $\delta\left(\mathbf{x},\mathbf{x}'\right)$ is the *Kronecker delta function* and $Q\left(\mathbf{x},\mathbf{x}'\right)$ is the corresponding kernel function that generates the kernel matrix defined in equation (4.39). $Q\left(\mathbf{x},\mathbf{x}'\right)$ can be explicitly defined as,

$$Q\left(\mathbf{x},\mathbf{x}'\right) = k\left(\mathbf{x},\mathbf{X}_u \mid \boldsymbol{\theta}\right)k^{-1}\left(\mathbf{X}_u,\mathbf{X}_u \mid \boldsymbol{\theta}\right)k\left(\mathbf{X}_u,\mathbf{x}' \mid \boldsymbol{\theta}\right) \quad (5.2)$$

According to equation (5.1), the kernel function in SPGP is actually a compound kernel based on the basic kernel $k\left(\mathbf{x},\mathbf{x}'\right)$. Generally $k\left(\mathbf{x},\mathbf{x}'\right)$ can be any valid kernel function such as the ARD kernel. Here $k\left(\mathbf{x},\mathbf{x}'\right)$ is parameterized by the hyper-parameters $\boldsymbol{\theta}$ and $k_{SPGP}\left(\mathbf{x},\mathbf{x}' \mid \mathbf{X}_u,\boldsymbol{\theta}\right)$ will depend on both $\boldsymbol{\theta}$ and the inducing vector $\mathbf{X}_u$. Please note that the basic kernel $k\left(\mathbf{x},\mathbf{x}'\right)$ is not directly parameterized by $\mathbf{X}_u$, as $\mathbf{X}_u$ are just input variables to it. Thus, if the basic kernel $k\left(\mathbf{x},\mathbf{x}'\right)$ is differentiable w.r.t. $\boldsymbol{\theta}$ and its input $\mathbf{x}$, the compounded kernel $k_{SPGP}\left(\mathbf{x},\mathbf{x}' \mid \mathbf{X}_u,\boldsymbol{\theta}\right)$ is also differentiable w.r.t. both $\boldsymbol{\theta}$ and $\mathbf{X}_u$.

To compute these derivatives, we reorganize the *training kernel matrix*, i.e., the kernel function $k_{SPGP}\left(\mathbf{x},\mathbf{x}' \mid \mathbf{X}_u,\boldsymbol{\theta}\right)$, as follows

$$\begin{aligned}
\mathbf{K}_{SPGP} &= k_{SPGP}\left(\mathbf{X},\mathbf{X} \mid \mathbf{X}_u,\boldsymbol{\theta}\right) \\
&= Q\left(\mathbf{X},\mathbf{X} \mid \mathbf{X}_u,\boldsymbol{\theta}\right) + \delta\left(\mathbf{X},\mathbf{X}\right)\left[k\left(\mathbf{X},\mathbf{X} \mid \boldsymbol{\theta}\right) - Q\left(\mathbf{X},\mathbf{X} \mid \mathbf{X}_u,\boldsymbol{\theta}\right)\right] \\
&= k\left(\mathbf{X},\mathbf{X}_u \mid \boldsymbol{\theta}\right)k^{-1}\left(\mathbf{X}_u,\mathbf{X}_u \mid \boldsymbol{\theta}\right)k\left(\mathbf{X}_u,\mathbf{X} \mid \boldsymbol{\theta}\right) \\
&\quad + \operatorname{diag}\left[k\left(\mathbf{X},\mathbf{X} \mid \boldsymbol{\theta}\right) - k\left(\mathbf{X},\mathbf{X}_u \mid \boldsymbol{\theta}\right)k^{-1}\left(\mathbf{X}_u,\mathbf{X}_u \mid \boldsymbol{\theta}\right)k\left(\mathbf{X}_u,\mathbf{X} \mid \boldsymbol{\theta}\right)\right] \\
&= \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf} + \operatorname{diag}\left[\mathbf{K}_{ff} - \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf}\right]
\end{aligned} \quad (5.3)$$

Here at the last step we denote $k\left(\mathbf{X},\mathbf{X}_u \mid \boldsymbol{\theta}\right)$, $k\left(\mathbf{X}_u,\mathbf{X}_u \mid \boldsymbol{\theta}\right)$ and $k\left(\mathbf{X}_u,\mathbf{X} \mid \boldsymbol{\theta}\right)$ shortly by $\mathbf{K}_{fu}$, $\mathbf{K}_{uu}$ and $\mathbf{K}_{uf}$, for the convenience of the following discussion.

To compute $\frac{\partial \mathbf{K}_{SPGP}}{\partial \boldsymbol{\theta}}$ and $\frac{\partial \mathbf{K}_{SPGP}}{\partial \mathbf{X}_u}$, we take a *divide-and-conquer* strategy by dividing $\mathbf{K}_{SPGP}$ into two parts and compute their derivatives separately. In more detail, based on equation (5.3), the *diagonal* and *non-diagonal* part of $\mathbf{K}_{SPGP}$ can be represented by,

$$diag\_part\left[\mathbf{K}_{SPGP}\right] = diag\_part\left[\mathbf{K}_{ff}\right] \tag{5.4}$$

$$nondiag\_part\left[\mathbf{K}_{SPGP}\right] = nondiag\_part\left[\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf}\right] \tag{5.5}$$

After the derivatives of these two parts are calculate separately, we can put them together to construct the derivatives of $\mathbf{K}_{SPGP}$. Because the corresponding evaluations of the derivatives involve heavy and complex matrix representations, we further simplify these computations by using a common trick that is widely used in deriving matrix derivatives—instead of directly computing the matrices' *derivatives*, we calculate their *differentials* first [Magnus and Neudecker 1999]. In general a differential and a derivative is connected by the definition,

$$d\mathbf{K} = \frac{\partial \mathbf{K}}{\partial \mathbf{t}} d\mathbf{t} \tag{5.6}$$

where $d\mathbf{K}$ is the differential and $\frac{\partial \mathbf{K}}{\partial \mathbf{t}}$ is the derivatives of the matrix w.r.t its vector parameter $\mathbf{t}$. The differentials of the diagonal and non-diagonal parts of $\mathbf{K}_{SPGP}$ are computed in the following tables.

Table 5-1: Deductions of the differentials of the training kernel matrix

1. Deduction of the differential of the diagonal part $d\left(diag\_part\left[\mathbf{K}_{SPGP}\right]\right)$

According to equation (5.4), the differential of the diagonal part of $\mathbf{K}_{SPGP}$ can be derived as,

$$\begin{aligned} d\left(diag\_part\left[\mathbf{K}_{SPGP}\right]\right) &= d\left(diag\_part\left[\mathbf{K}_{ff}\right]\right) \\ &= diag\_part\left[d\mathbf{K}_{ff}\right] \end{aligned} \tag{5.7}$$

where $d\mathbf{K}_{ff}$ depends on $\frac{\partial \mathbf{K}_{ff}}{\partial \boldsymbol{\theta}}$ and $\frac{\partial \mathbf{K}_{ff}}{\partial \mathbf{X}_u}$, which will be computed in Table 5-2.

2. Deduction of the differential of the non-diagonal part $d\left(nondiag\_part\left[\mathbf{K}_{SPGP}\right]\right)$

Based on equation (5.5), the differential of the non-diagonal part can be evaluated as,

$$\begin{aligned} d\left(nondiag\_part\left[\mathbf{K}_{SPGP}\right]\right) &= d\left(nondiag\_part\left[\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf}\right]\right) \\ &= nondiag\_part\left[d\left(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf}\right)\right] \end{aligned} \tag{5.8}$$

where $d\left(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf}\right)$ can be further extended as follows,

$$\begin{aligned} & d\mathbf{vec}\left(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf}\right) \\ &= \mathbf{vec}\left(d\left(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf}\right)\right) && (1) \\ &= \mathbf{vec}\left(\left(d\mathbf{K}_{fu}\right)\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf} - \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\left(d\mathbf{K}_{uu}\right)\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf} + \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\left(d\mathbf{K}_{uf}\right)\right) && (2) \\ &= \mathbf{vec}\left(\left(d\mathbf{K}_{fu}\right)\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf}\right) - \mathbf{vec}\left(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\left(d\mathbf{K}_{uu}\right)\mathbf{K}_{uu}^{-1}\mathbf{K}_{uf}\right) \\ &\quad + \mathbf{vec}\left(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\left(d\mathbf{K}_{uf}\right)\right) && (3) \\ &= \left(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1} \otimes \mathbf{I}\right)\mathbf{vec}\left(d\mathbf{K}_{fu}\right) - \left(\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1} \otimes \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\right)\mathbf{vec}\left(d\mathbf{K}_{uu}\right) \\ &\quad + \left(\mathbf{I} \otimes \mathbf{K}_{fu}\mathbf{K}_{uu}^{-1}\right)\mathbf{vec}\left(d\mathbf{K}_{uf}\right) && (4) \end{aligned} \tag{5.9}$$

Here $\otimes$ is the *Kronecker product* on matrices and $\mathbf{vec}\left(\mathbf{X}\right)$ is the *vectorization* of the matrix $\mathbf{X}$ formed by stacking the columns of $\mathbf{X}$ into a single column vector.

In the derivation of equation (5.9), step (1) is directly based on equation (11) in appendix and step (2) is based on equation (10). Step (3) and (4) are based on equation (12) in appendix and the kernel function's symmetric property that implies the following relationship,

$$\begin{aligned}\mathbf{K}_{fu} &= \mathbf{K}_{uf}^{T} \\ \mathbf{K}_{uu} &= \mathbf{K}_{uu}^{T}\end{aligned} \tag{5.10}$$

In the final representation, $d\mathbf{K}_{fu}$, $d\mathbf{K}_{uu}$, and $d\mathbf{K}_{uf}$ will depend on their corresponding derivatives with respect to $\boldsymbol{\theta}$ and $\mathbf{X}_u$, which are given in Table 5-2. The calculation of $\mathbf{K}_{uu}^{-1}$ can be done by *Cholesky decomposition* (see appendix A.4 in [Rasmussen and Williams 2006]). As $\mathbf{K}_{fu}\mathbf{K}_{uu}^{-1} \otimes \mathbf{I}$ usually results in a very sparse matrix and some tactics can be exploited accordingly to speed up its computation.

As discussed in the deduction, the evaluations in Table 5-1 depend on $d\mathbf{K}_{ff}$ $d\mathbf{K}_{fu}$, and $d\mathbf{K}_{uu}$, and $d\mathbf{K}_{uf} = d\mathbf{K}_{fu}^{T}$ according to symmetry. We compute them by giving their derivatives w.r.t. the corresponding parameters in Table 5-2.

Table 5-2: Deductions of the corresponding derivatives used in Table 5-1

1. Derivatives with respect to hyper-parameters $\boldsymbol{\theta}$

The Deductions of $\frac{\partial \mathbf{K}_{ff}}{\partial \boldsymbol{\theta}}$, $\frac{\partial \mathbf{K}_{fu}}{\partial \boldsymbol{\theta}}$, $\frac{\partial \mathbf{K}_{uu}}{\partial \boldsymbol{\theta}}$ are all straightforward. In fact, they all take the same form,

$$\frac{\partial \mathbf{K}}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial k_{11}}{\partial \boldsymbol{\theta}} & \cdots & \frac{\partial k_{1n}}{\partial \boldsymbol{\theta}} \\ \vdots & \ddots & \vdots \\ \frac{\partial k_{m1}}{\partial \boldsymbol{\theta}} & \cdots & \frac{\partial k_{mn}}{\partial \boldsymbol{\theta}} \end{pmatrix} \text{ where } \mathbf{K} = \mathbf{K}_{ff}, \mathbf{K}_{fu} \text{ or } \mathbf{K}_{uu}. \tag{5.11}$$

Here the (i,j) entry $\frac{\partial k_{ij}}{\partial \boldsymbol{\theta}}$ is the derivative of the kernel function evaluated on the corresponding inputs, i.e., $\frac{\partial k(\mathbf{x}_i, \mathbf{x}_j \mid \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$. The value of $\frac{\partial k_{ij}}{\partial \boldsymbol{\theta}}$ depends on the specific form of the kernel function used.

2. Derivatives with respect to inducing vectors $\mathbf{X}_u$

The matrices' derivatives with respect to $\mathbf{X}_u$ are different from those evaluated above. In fact, these matrices depend on $\mathbf{X}_u$ in very different ways, e.g., $\mathbf{K}_{ff}$ doesn't even depend on $\mathbf{X}_u$ at all. As a result, their derivatives are also evaluated differently. For the convenience of our discussion, we first denote the *i*th instance in the inducing vectors $\mathbf{X}_u$ by $\mathbf{x}_{u(i)}$, its *d*th component by $x_{u(i)}^d$, and the *j*th training case in the training set $\mathbf{X}$ by $\mathbf{x}_j$. Thus, $x_{u(i)}^d$ actually corresponds to the (i,d) entry in the $\mathbf{X}_u$ matrix, and $\mathbf{x}_j$ corresponds to the *j*th row in $\mathbf{X}$.

With these notations, the derivatives of the corresponding matrices can be represented by the follows.

$$\frac{\partial \mathbf{K}_{ff}}{\partial x_{u(i)}^d} = \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix} \tag{5.12}$$

$$\frac{\partial \mathbf{K}_{fu}}{\partial x_{u(i)}^{d}} = \begin{pmatrix} 0 & \cdots & \frac{\partial k\left(\mathbf{x}_1, \mathbf{x}_{u(i)}\right)}{\partial x_{u(i)}^{d}} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{\partial k\left(\mathbf{x}_j, \mathbf{x}_{u(i)}\right)}{\partial x_{u(i)}^{d}} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \underbrace{\frac{\partial k\left(\mathbf{x}_N, \mathbf{x}_{u(i)}\right)}{\partial x_{u(i)}^{d}}}_{i\text{th column}} & \cdots & 0 \end{pmatrix} \tag{5.13}$$

$$\frac{\partial \mathbf{K}_{uu}}{\partial x_{u(i)}^{d}} = \begin{pmatrix} 0 & \cdots & \frac{\partial k\left(\mathbf{x}_{u(1)}, \mathbf{x}_{u(i)}\right)}{\partial x_{u(i)}^{d}} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\partial k\left(\mathbf{x}_{u(i)}, \mathbf{x}_{u(1)}\right)}{\partial x_{u(i)}^{d}} & \cdots & 0 & \cdots & \left.\frac{\partial k\left(\mathbf{x}_{u(i)}, \mathbf{x}_{u(M)}\right)}{\partial x_{u(i)}^{d}}\right\} i\text{th row} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \underbrace{\frac{\partial k\left(\mathbf{x}_{u(M)}, \mathbf{x}_{u(i)}\right)}{\partial x_{u(i)}^{d}}}_{i\text{th column}} & \cdots & 0 \end{pmatrix} \tag{5.14}$$

In all above equations $k(\mathbf{x}, \mathbf{x}')$ corresponds to the basic kernel, e.g., an ARD kernel that will be discussed in Section 5.1.2. In equation (5.12), the derivatives of $\mathbf{K}_{ff}$ are all zeros because $\mathbf{K}_{ff}$ doesn't explicitly depend on $\mathbf{X}_u$. In equation (5.13), $\frac{\partial \mathbf{K}_{fu}}{\partial x_{u(i)}^{d}}$ only have non-zero entries on its ith column. Accordingly, $\frac{\partial \mathbf{K}_{uf}}{\partial x_{u(i)}^{d}}$ are only non-zero on its ith row. As for $\frac{\partial \mathbf{K}_{uu}}{\partial x_{u(i)}^{d}}$, the non-zero entries will be on both the ith row and the ith column, except that the (i,i) entry is again 0.

The details of the Deductions of $\frac{\partial k\left(\mathbf{x}_1,\mathbf{x}_{u(i)}\right)}{\partial x_{u\,(i)}^{d}}$ or $\frac{\partial k\left(\mathbf{x}_{u(i)},\mathbf{x}_{u(i)}\right)}{\partial x_{u\,(i)}^{d}}$ will depend on the specific kernel function $k(\mathbf{x},\mathbf{x}')$, which is usually straightforward in practice.

So far we have developed the requisite calculations $\frac{\partial \mathbf{K}_{SPGP}}{\partial \boldsymbol{\theta}}$ and $\frac{\partial \mathbf{K}_{SPGP}}{\partial \mathbf{X}_u}$ that enable us to apply the Laplace classification framework to the SPGP model. The classification model is called the *SPGP Classifier* (SPGPC). The training of the model includes optimizing both $\boldsymbol{\theta}$ and $\mathbf{X}_u$ by maximizing the marginal likelihood (equation (4.51)). The detail about making new predictions by SPGPC and optimizing the hyperparameters $\boldsymbol{\theta}$ and $\mathbf{X}_u$ through marginal likelihood can be found in Section 4.1.3 and 4.1.4.

The above extension work increases the computation complexity, because now the model's computation cost is susceptive to both the instance size and the input dimensionality. However, with the assistance of an ARD kernel, it turns out that this computation cost can be kept in hand in practice. In fact, the SPGPC model with an ARD kernel can bring us a dual sparseness, that is, the sparseness in both the original input space and the kernel space. We will explore this idea in more detail in the next section.

5.1.2 ARD Kernel and Feature Selection

In the above extensions, although any valid kernel function can be uses as the basic kernel function $k(\mathbf{x},\mathbf{x}')$, an ARD kernel is used here in SPGPC because it can generates feature selection results and thus reduces the extra computation costs incurred in SPGPC. In Section

4.1.1, we have briefly introduced the ARD kernel. Recall that the ARD kernel function is represented by,

$$k_{ARD}(\mathbf{x},\mathbf{x}') = a^2 \exp\left[-\frac{1}{2}\sum_{d=1}^{D}\left(\frac{x_d - x_d'}{\lambda_d}\right)^2\right] \tag{5.15}$$

Here the function is parameterized by the hyper-parameters $\boldsymbol{\theta} = \left[a^2, \left\{\lambda_d^2\right\}_{d=1}^{D}\right]^T$. Among these parameters, a^2 represents the *signal attitude* and λ_d is the *characteristic length-scale* on the *d*th dimension. Intuitively, λ_d represents how far the input needs to move along the *d*th dimension for the function values to become uncorrelated. Accordingly, the inverse of the length scale can be used to determine how relevant an input is—if the length scale has a very large value, the output covariance (i.e., the kernel value) will become almost independent of that input, which effectively removes it from the inference. Thus, filtering out the inputs with a large length-scale value essentially correspond to a process of feature selection. As previously mentioned, the appropriate values of these λ_d are usually determined by the maximization of the marginal likelihood.

Using an ARD kernel can provide extra information about the problem structure as it is able to give a ranking of input features. It also plays an important role in reducing the computation complexity of the SPGPC model. In general, computing the derivatives of the matrix $\mathbf{K}_{SPGP}$ generally takes $\mathrm{O}\left(M^2N + MND\right)$ time, where N is the number of training instances, M is the number of inducing variables and D is the input dimensionality. Although SPGPC can reduce this time complexity by decreasing the number of inducing variables M, the model will still suffer the curse of dimensionality when the number of input dimensions D is high. However, by selecting the most relevant features according to the results from the ARD kernel, the number D can be effectively reduced as well. We will use this technique in the experiments in Section 5.3.

5.2 Connections between ENRBFC and SPGPC

It is interesting to compare the two different classification models that we have proposed in this thesis. The difference between ENRBFC and SPGPC essentially reflects the difference between the traditional parametric models and the Bayesian nonparametric models. As discussed in Section 2.1.3, the appropriate choice of the two models heavily depends on the specific learning tasks. For example, because their respective computation costs are $O(NP)$ and $O(NM^2)$ respectively, the computation cost of ENRBFC is usually smaller than SPGPC in a moderate-dimensional and large-scale learning problem. Here N is the training set size, P is the number of iterations in the EM algorithm, and M is the number of inducing vectors. Generally $O(NP) < O(NM^2)$ for large scale data sets where both N and M is much bigger than P. On the other hand, the results of SPGPC usually provide richer information than ENRBFC as it can generate fully probabilistic predictions. Other practical considerations include that ENRBFC is usually easier to implement, whereas SPGPC is able to provide a ranking of input features with the assistance of specific kernels.

There is also a deep connection between the two models. To see this, we rewrite the expression for the predictive mean of SPGP in Equation (4.49) as,

$$\begin{aligned}
E\left[q_{SPGP}\left(\mathbf{f}_* \mid \mathbf{y}\right)\right] &= \mathbf{Q}_{*f}\left(\mathbf{Q}_{ff}+\mathbf{\Lambda}\right)^{-1}\mathbf{y} \\
&= \mathbf{K}_{*u}\left(\mathbf{K}_{uu}+\mathbf{K}_{uf}\left(\mathbf{\Lambda}+\sigma^2\mathbf{I}\right)^{-1}\mathbf{K}_{fu}\right)^{-1}\mathbf{K}_{uf}\left(\mathbf{\Lambda}+\sigma^2\mathbf{I}\right)^{-1}\mathbf{y} \\
&= \mathbf{K}_{*u}\mathbf{w} \qquad (5.16) \\
&= \sum_{j=1}^{M} k\left(\mathbf{x}_*, \mathbf{x}_j\right) w_j \\
&= \sum_{j=1}^{M} \phi_j\left(\mathbf{x}_*\right) w_j
\end{aligned}$$

By viewing the kernel functions $k\left(\mathbf{x}_*, \mathbf{x}_j\right)$ evaluated on the *j*th inducing vector $\mathbf{x}_j$ as a basis function, and the complex term $w_j = \left[\left(\mathbf{K}_{uu}+\mathbf{K}_{uf}\left(\mathbf{\Lambda}+\sigma^2\mathbf{I}\right)^{-1}\mathbf{K}_{fu}\right)^{-1}\mathbf{K}_{uf}\left(\mathbf{\Lambda}+\sigma^2\mathbf{I}\right)^{-1}\mathbf{y}\right]_j$ as the corresponding coefficient of the basis function, we can see there is a correspondence between equation (5.16) for SPGP and equation (2.16) for RBF. Because SPGP is a full probabilistic model, it is actually a Bayesian version of RBF constructed by placing a Gaussian prior on the coefficients [Bishop 2006; Snelson 2007]. The only difference is that the Bayesian RBF constructed this way will lead to a degenerate GP approximation like the SR model (Section 4.3.3), whereas SPGP is not degenerate.

In view of this connection between RBF and SPGP, the SPGPC model can be viewed as a special Bayesian RBF based classifier. Moreover, as discussed in Chapter 3, the ENRBFC model can also be viewed as a special extension of the RBF model. Thus, both SPGPC and ENRBFC can actually viewed as different kinds of extensions based on the traditional RBF model. This deep similarity reflects the general relations between parametric and nonparametric models, as stated in *Mercer's theorem* [Konig 1986].

5.3 Experimental Results and Analysis

In this section we compare the proposed SPGPC model with some other sparse GP classifiers. These classifiers are based on the extension of the full GP, SD, SR, and PP models that are introduced in Section 4.3. They are extended from the corresponding regression models based on the similar techniques as used to derive SPGPC. The classification problems used in the experiments are from both benchmarks and real-world applications.

Compared to the other sparse GP classifiers, SPGPC has several advantages that are inherited from the regression SPGP model, such as the extra flexibility in optimizing the inducing vectors and the avoidance of anti-intuitive and inconsistent results appearing in SR and PP (refer to Section 4.3.6 for details). As a result, the learning accuracy of SPGPC is usually better than the other sparse GP classifiers, and the number of inducing vectors used by it is usually smaller, which in turn results in a time reduction at both the training and the evaluation stages.

The specifications of the sparse GP classifiers used in the experiments are summarized as follows.

1. *Selection of the Basic Kernel Function*: the ARD type kernel [Neal 1996] that is described in equation (5.15) is used for all classifiers. The ARD kernel generates useful feature selection results, which helps alleviate the curse of dimensionality issue. As the ARD kernel is sensitive to the value ranges of the inputs, we always standardize the input data into zero mean and unit variance.

2. *Selection of the GP Classification Framework*: The general classification framework used here is the *Laplace Approximation* [Rasmussen and Williams 2006], as introduced in Section 4.1.3 and 4.1.4. Compared to other popular frameworks such as

Expectation Propagation, it is easier to implement in practice. In this framework we use the *logistic* function as the squashing function.

3. *Determination of the Inducing Vectors and Kernel Hyper-parameters*: As proposed in Section 5.1, in SPGPC both types of parameters are optimized in a seamless computing process by maximizing the marginal likelihood. In the other sparse models, these two types of parameters are optimized separately. For the sake of implementation simplicity, the inducing vectors are selected as a random subset of training data in these models. And to avoid any possible loss in accuracy caused by a specific initialization, several random initializations are tried in each experiment. As for the optimization of the kernel hyper-parameters, we follow the strategy exploited in the experiment study in [Snelson 2007]. In details, the full GP and the SD model optimize the kernel hyper-parameters through the maximization of marginal likelihood. The SR/PP models use the same optimal values as them.

4. *Optimization Method*: The *Conjugate Gradient* method is used in the experiments as a general optimization method. There is one of its implementation that is available online at http://www.kyb.tuebingen.mpg.de/bs/people/carl/code/minimize/.

5. *The Results of the SR and PP Classifier*: As discussed in Section 4.3.3 and 4.3.4, the predictive distributions of SR and PP are exactly the same except the difference between their predictive variances. However, as far as a classification problem is concerned, the classification result will only depend on the mean of the predictive distribution that gives the posterior probability of a specific class label. Thus, in the experiments we will put the results of SR and PP classifiers together to compare with other models.

In the following subsections, through a series of experimental analysis, we will compare the different models in terms of their *classification rates* (accuracy), *training times*, and *evaluation times*.

5.3.1 Artificial Binary Classification

In this section, the same 2D classification dataset in Section 3.3.1 is used to give a bird-eye view of different models. We have used this problem to show the different decision boundaries generated from several classifiers against the optimal Bayesian one. To summarize, the data are generated from a mixture of three Gaussian components, of which one component is labeled as class 1 and the other two are labeled as class 2. The proportions of the two classes are equal. The Bayesian decision boundary can be generated as we have already known the real distribution of the data. By comparing the decision boundaries from different models with the Bayesian one, the *classification performances* of the classifiers can be evaluated.

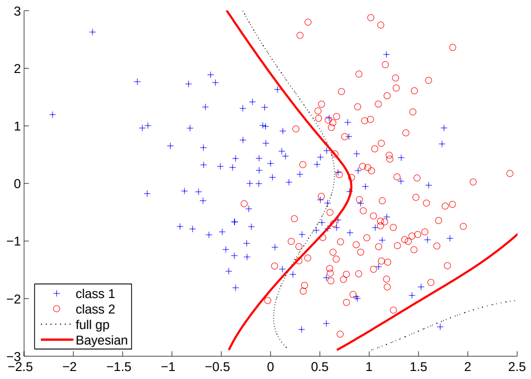


Figure 5-1: Decision boundary generated by Full GP Classifier

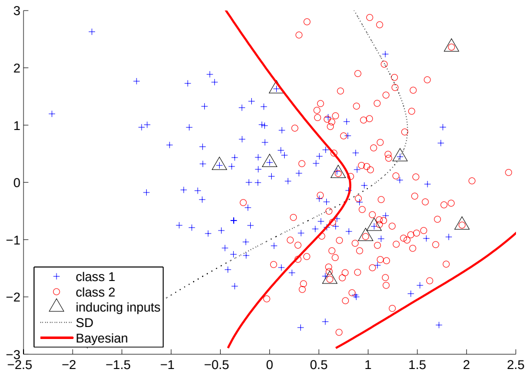


Figure 5-2: Decision boundary generated by SD Classifier

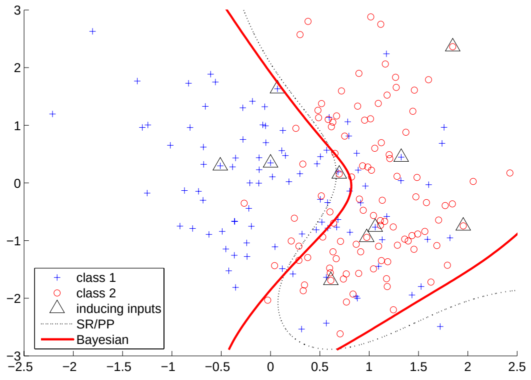


Figure 5-3: Decision boundary generated by SR/PP Classifier

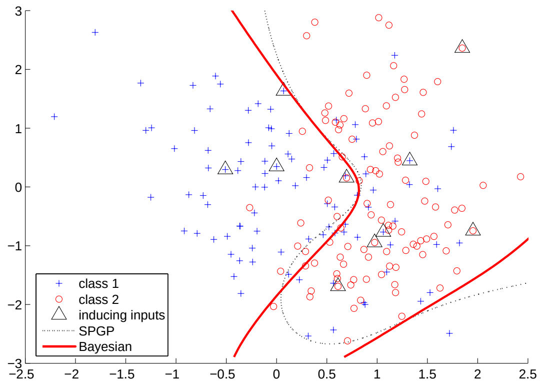


Figure 5-4: Decision boundary generated by SPGP Classifier

The decision boundaries generated by different models are plotted in Figure 5-1~Figure 5-4. Here one of our intentions is to demonstrate the *approximating ability* of the different sparse models to the full GP. Thus, the same fixed set of inducing vectors is used in all the sparse models, plotted as the triangles in Figure 5-2~Figure 5-4. These inducing vectors are randomly selected from the training cases. In the figures, the solid line is the Bayesian decision boundary, and the dotted line is the one from the corresponding classifier's result.

From the comparison we can see that most of the sparse models are able to generate a decision boundary close to the Bayesian one, except there is an obvious gap in SD's result. Moreover, the decision boundaries generated by SR, PP and SPGPC are very similar to the one by the full GP, whereas only 10 inducing vectors are used in these models (compared to 200 vectors in the training set). The poor performance of SD can be explained by its ignorance the training cases other than the inducing vectors.

To make the experiment more interesting, we test the ARD kernel function in a feature selection scenario, by augmenting the original two inputs with four extra input variables. We denote the six features as x_1 to x_6. Among the four newly added features x_3 to x_6, x_3 and x_4 are the copies of the original two inputs (x_1 and x_2), interrupted by an additive Gaussian noise with zero mean and a deviation of 0.02. The inputs x_5 and x_6 are pure noises generated by a standard Gaussian. Our interest here is to see if the ranking generated by the ARD kernel is able to filter out the unrelated inputs x_5 and x_6. Optimized by the conjugate gradient method, the optimal length scales for the six features are given in Table 5-3. The results are from SPGPC model and those generated by the other models give very similar values.

Table 5-3: Length scales of different input features in an ARD kernel

Input Features	x_1	x_2	x_3	x_4	x_5	x_6
Length Scales	.57921	1179.6	332.54	1399.1	4683.9	1966.2

As we discussed in Section 5.1.2, the relative importance of the features can be measured by the inverse of the corresponding length scales. Therefore, the ranking of the features beginning from the most important to the least is x_1, x_3, x_2, x_4, x_6, and x_5. Here we can see that SPGPC successfully picks out the unrelated features as x_5 and x_6. However, at the first glance it seems a little surprising that input x_3 is ranked as more important than x_2. Further experiments reveal the fact that by using one feature alone, x_1 achieves 74.0% classification rate, x_3 gives 73.5%, whereas x_2 gives only 58.0%. Thus, because in the original problem x_1 is more significant than x_2 and the noise level interrupting x_3 and x_4 is not very high, it turns out that x_3 still remains more significant than x_2. In fact we can see from the above figures that the changing direction of the instances' class labels almost comes along with the x-axis, which corresponds to x_1. By using an ARD kernel, the SPGPC model is capable of capturing this structure in the problem.

5.3.2 E.Coli Protein Localization Site

The second problem is from the UCI machine learning repository (online available at http://mlearn.ics.uci.edu/MLSummary.html), which uses the gene data to predict the localization site of the E.coli protein. The dataset consists of 336 instances that can be classified into 8 different location sites. There are totally 7 input features used in the experiment, excluding the original eighth "sequence name" feature because it bears no significant connection to the prediction. The original data set are used for both training and testing. Thanks to marginal likelihood method in GP's model selection (discussed in Section 4.1.1 and 4.1.4), using the same data as both training and testing will not lead to model's over-fitting in this case. Generally, this is one of the advantages of the GP models over the parametric methods, because the relatively small data set can be reused this way.

Different from the first experiment, the problem here is a *multi-classification* problem. However, the Laplace approximation that we used in the extension work is generally used for binary classification purposes. To overcome this limitation, a widely used multi-classification strategy called *one-versus-the-rest* [Bishop 2006] is used in the experiment. In essence, this strategy solves a *k*-classification problem by dividing it into *k* binary classifications. In each binary classification, a separate learning model is fitted to discriminate one class from the others. Although there are some practical difficulties with this method, it has been widely used to apply the binary-classification models to the multi-classification problems. Here another advantage of this method is that by using an ARD kernel, we can find separate sets of useful features for each individual class. This technique will also be used to solve other multi-classification problems in the following experiments.

Table 5-4: Classification Results of Different Models on Ecoli Dataset

Inducing set size		4	8	16	32	64	128	256
Accuracy(%)	Full GP	90.18%						
	SD	46.73	71.43	70.54	77.68	82.44	88.39	90.18
	SR/PP	46.13	70.54	81.55	84.23	87.80	87.20	91.07
	SPGPC	80.65	84.52	84.52	86.31	90.77	90.48	93.75
Training Time (s)	Full GP	165.85						
	SD	2.99	2.81	4.35	5.31	8.11	19.98	92.56
	SR/PP	6.71	6.82	6.83	6.94	9.32	15.12	57.78
	SPGPC	4.36	5.00	6.51	10.66	19.76	47.54	190.71
Evaluation Time (s)	Full GP	2.15						
	SD	0.04	0.04	0.05	0.08	0.12	0.27	1.2
	SR/PP	0.66	.077	0.78	0.79	0.82	0.81	0.84
	SPGPC	≈ 0	≈ 0	≈ 0	≈ 0	0.01	0.01	0.03

The results from different models are summarized in Table 5-4, in which the accuracy (classification rate), training time and evaluation time of each model are compared. As for the sparse models, their *model complexities* are effectively controlled by the number of inducing vectors. Generally, when more inducing vectors are used, their approximating abilities increase, at a cost of prolonged training and evaluation times.

Compared to other sparse GP classifiers, SPGPC gives a better accuracy here although the number of inducing vectors in all sparse models is the same. In fact, with only 64 inducing vectors, SPGPC is able to achieve a comparable classification rate as the full GP, whereas this number for other sparse models is roughly 256. As a result, both the training time and evaluation time in SPGPC (i.e., 19.76s and 0.01s respectively) have been much shortened, although for a fixed number of inducing vectors it takes a longer time than others due to the extra calculations of the parameter derivatives. The better performance of SPGPC can be explained by its advantages as discussed in Section 4.3.6.

In spite of the small differences, the feature selection results from different models are basically in accordance with one another. In summary, the features *alm1* and *alm2* are important to the predictions of all location sites. With these two features alone, a full GP gives a classification rate of 75.06%. It is also observed that the *mcg* attribute is closely connected to the prediction of the location *imU*, the *acc* attribute is connected to location *om*, and the attributes *mcg* and *gvh* is connected to location *pp*.

5.3.3 EEG Based Emotion Recognition

The experiment in this section is from a real-world classification problem, where the Electroencephalography (EEG) signals are used to identify the emotions of a subject. In recent years the BCI (brain computer interface) data have been widely used in different types of machine learning applications [Takahashi 2000; Geng, Gan et al. 2008; Zhou, Gan et al. 2008]. Generally the researchers are interested not only in detecting the brain activities but also in selecting the useful features that can be extracted from the original data. In this experiment the data are collected in a private laboratory. With the assistance of several kinds of visual and aural stimuli, the participants excited three types of emotions, namely *happiness*, *relaxation*, and *sadness*. The EEG signals are recorded from 4 picked channels, namely F4, T3, T4 and P4 as referred to the standard 10-20 EEG system that is described in Figure 5-5. To increase the prediction accuracy, six statistical features are extracted from the recorded signal of each channel. This idea is borrowed from [Takahashi 2000] and the numeric features are listed in Table 5-5. Thus, there are totally 24 input features and 3 types of emotions to predict. After the extraction of features, there are totally 360 numeric instances generated from the original signals, each of which is several seconds long in the original signal.

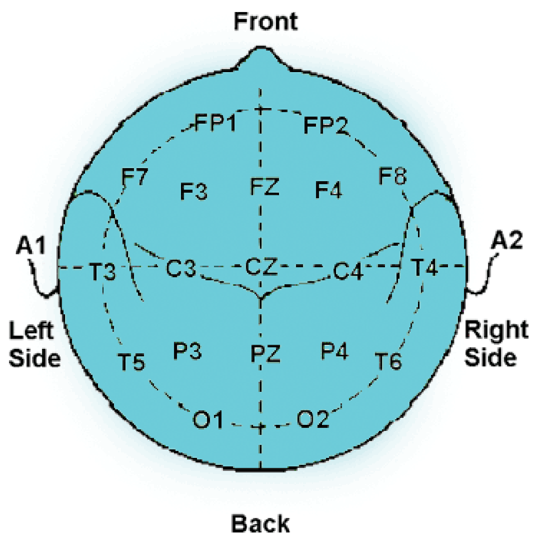


Figure 5-5: EEG 10-20 system: in the experiment the 4 channels are picked as F4 (channel 1), T3 (channel 2), T4 (channel 3), and P4 (channel 4). The ground and reference are selected as A1 and FP2 respectively.

Table 5-5: Six numeric features that are extracted from the original EEG signals

$$\mu_X = \frac{1}{T}\sum_{t=1}^{T} X(t)$$

$$\sigma_X = \sqrt{\frac{1}{T}\sum_{t=1}^{T}\left(X(t)-\mu_X\right)^2}$$

$$\delta_X = \frac{1}{T-1}\sum_{t=1}^{T-1}\left|X(t+1)-X(t)\right|$$

$$\bar{\delta}_X = \frac{1}{T-1}\sum_{t=1}^{T-1}\left|\bar{X}(t+1)-\bar{X}(t)\right|$$

$$\gamma_X = \frac{1}{T-2}\sum_{t=1}^{T-2}\left|X(t+2)-X(t)\right|$$

$$\bar{\gamma}_X = \frac{1}{T-2}\sum_{t=1}^{T-2}\left|\bar{X}(t+2)-\bar{X}(t)\right|$$

Table 5-6: Classification Results of Different Models on EEG Emotion Recognition

Inducing set size		4	8	16	32	64	128	256
Accuracy (%)	Full GP	100%						
	SD	52.22	53.89	63.06	74.17	75.56	94.44	99.44
	SR/PP	56.39	66.94	80.83	86.39	85.28	95.00	100
	SPGPC	52.78	68.89	87.50	87.78	94.44	93.89	98.89
Training Time (s)	Full GP	207.90						
	SD	1.56	1.58	1.84	2.13	4.49	13.07	78.70
	SR/PP	3.99	3.71	2.00	2.15	2.54	5.82	31.21
	SPGPC	2.28	2.44	3.83	6.17	14.20	30.58	122.19
Evaluation Time (s)	Full GP	1.9						
	SD	0.02	0.02	0.03	0.03	0.06	0.18	1.11
	SR/PP	0.60	0.61	0.67	0.68	0.68	0.69	2.63
	SPGPC	≈ 0	≈ 0	≈ 0	≈ 0	0.01	0.01	0.04

The classification results and the corresponding training/evaluation times of different models are compared in Table 5-6. Here the *one-versus-the-rest* strategy [Bishop 2006] is used for multi-classification. From the results we can see that with only 128 instances (less than the half of the training set), the sparse models have been able to achieve over 90% accuracy. This brings the significant time reductions in those sparse models. Here the advantage of SPGPC over other sparse models is not as much as in the previous experiments. This is because compared to the previous classification problems, the problem here is relatively simple (we will see in the follows that the prediction is dominated by only one or two channels). However, it can still be roughly observed that SPGPC has a faster improving rate of accuracy along the increase of the inducing set size. The main disadvantage of SPGPC is still its relatively longer training time compared to other sparse models when the number of inducing vectors is fixed. However, the evaluation of SPGPC is usually faster because its matrix inversion in equation (4.49) is simpler than the other sparse models.

The feature ranking results from SPGPC reveals that Channel T4 has the most significant influence on determining all emotion types. By using the six statistical features extracted from T4 alone, a 90.83% overall classification rate can be achieved. Besides T4, Channel T3 also plays an important role in differentiating *happiness* with the two type types.

5.3.4 Spam Email Identification

Through the above experiments we have seen the advantages of the sparse GP classifiers over the full GP in achieving a comparable accuracy while with much reduced training/evaluation times. To further test these models, some large-scale classification problems will be used in the following two experiments. These problems are more difficult in the sense that there are more input features involved and the training data size is larger.

The experiment presented in this section is the *spam email identification* dataset from UCI Machine Learning Repository. There are totally 57 numeric features involved, including various statistic parameters from both the normal and spam email samples, e.g., the frequency of a specific word or punctuation. The goal is to identify if an instance is a spam email. Among the whole data, we use 3101 instances as training data and the rest 1500 for test purpose. The classification rates and time differences are given in Table 5-7.

Table 5-7: Classification Results on Spam Email Identification

Inducing set size		4	8	16	32	64	128	256	512	1024	2048
Accuracy (%)	Full GP	95.53%									
	SD	61.60	65.27	62.39	74.67	75.93	85.60	89.00	90.00	92.40	94.20
	SR/PP	64.60	81.67	76.87	85.33	85.13	82.53	88.87	90.40	89.80	91.73
	SPGPC	89.80	91.00	91.60	92.93	92.60	92.67	93.73	92.47	93.87	94.27
Training Time (s)	Full GP	10539.4									
	SD	1.10	1.03	1.14	1.15	1.9	6.19	31.34	153.16	931.85	4364.06
	SR/PP	3.41	3.60	3.47	3.58	4.53	6.31	28.03	117.07	698.50	3289.93
	SPGPC	6.39	10.02	17.94	40.44	87.93	219.30	476.57	1158.80	3155.76	9943.62
Evaluation Time (s)	Full GP	76.44									
	SD	0.02	0.02	0.03	0.03	0.05	0.11	0.41	1.62	8.53	36.21
	SR/PP	15.53	14.83	18.57	18.75	16.57	27.70	18.61	16.16	16.00	18.40
	SPGPC	0.01	0.01	0.01	0.02	0.04	0.08	0.18	0.45	1.18	3.53

In the result we highlight that with only 8 instances the SPGPC model is able to achieve a classification rate over 90%, whereas accuracy of the full GP (trained on over 3000 instances) is around 95%. With 8 instances, the training time of SPGPC is only 10.02 seconds, in contrast to 10539.4 seconds (about 2.9 hours) used by the full GP. Thus, at a cost of a small loss of accuracy, the SPGPC model achieves significant reductions in time complexity. Similar reductions can also be obtained in other sparse models, although not as much as those in SPGPC.

In practice, it is even more important to find out which features are most related to the reorganization of a spam email. We find the most related features by optimizing ARD kernel. Different from the previous experiments where the feature ranking results from different models are close to each other; here the diversity dominates as the number of input features involved here is larger. Accordingly, a *voting* strategy is used to combine the different ranking results from different models. In details, separate sets of significant features are first selected based on the individual ranking results from each model. Then the features that have been selected by at least two learning models will be further picked out to form the final set of significant features. The 13 significant features that are generated this way are,

word_freq_our, *word_freq_remove*, *word_freq_receive*, *word_freq_money*, *word_freq_hp*, *word_freq_george*, *word_freq_lab*, *word_freq_telnet*, *word_freq_technology*, *word_freq_original*, *word_freq_edu*, *word_freq_conference*, *char_freq_$*.

Among the rest, these 13 features alone achieve a classification rate of 89.67%. They correspond to the most related words or punctuation in differentiating a spam email from a normal one.

5.3.5 USPS Hand-written Digit Classification (Digit 3 VS Digit 5)

The last experiment is a well-known image classification problem—the USPS handwritten image recognition problem [Rasmussen and Williams 2006] (online available at http://www.gaussianprocess.org/). Here we will focus on the classification of digit 3 and 5, for an illustration purpose. Similar to the previous experiment, it is a high-dimensional large-scale classification problem. The input features are the 256 pixel densities of the image. There are 767 cases in the training set and 773 in the test.

Table 5-8: Classification Results on USPS hand-written Digit Classification (digit 3 v.s. digit 5)

Inducing set size		4	8	16	32	64	128	256	512
Accuracy (%)	Full GP	96.51%							
	SD	46.05	47.09	54.20	59.38	73.61	91.46	94.57	94.83
	SR/PP	48.90	69.34	69.60	78.40	78.14	87.71	96.77	96.64
	SPGPC	78.01	79.17	83.05	87.97	91.59	93.40	97.54	96.38
Training Time (s)	Full GP	3331.94							
	SD	2.61	1.85	2.07	2.38	15.20	33.50	213.97	1053.27
	SR/PP	4.81	7.75	4.74	10.75	28.53	71.96	221.72	913.71
	SPGPC	8.90	11.21	17.55	29.73	65.85	180.91	466.20	1479.76
Evaluation Time (s)	Full GP	11.24							
	SD	0.07	0.07	0.08	0.1	0.14	0.27	1.24	5.38
	SR/PP	1.38	1.37	1.69	5.05	5.51	11.67	11.74	11.58
	SPGPC	0.01	0.02	0.02	0.02	0.03	0.06	0.11	0.27

The results from different models are summarized in Table 5-8. With at most 256 instances, all the sparse models can achieve an accuracy that is comparable to the one by full GP. On the other hand, the training and evaluation times in these models have been reduced significantly. And this reduction in SPGPC is even more. It is observed that with only 4 instances, the accuracy of SPGPC has already been 78.01%. Here the advantages of sparseness have been fully exploited in SPGPC, due to its flexibility provided by determining the position of inducing vectors optimally.

Based on the corresponding length scales of the ARD kernel, a ranking of different input features (i.e., different pixels) can be generated. For example, in SPGPC, by using 512 inducing vectors, 105 pixels selected by optimizing ARD can give an accuracy of 95.34%, and the training time has been reduced from 1479.76 seconds (with all 256 features) to 1250.11 seconds. By setting a higher importance threshold, 37 features can be further picked out, which result in an even higher accuracy of 96.12%, with a training time reduced to

479.91 seconds. We contrast the selected 37 features against a sample of digit 3 and 5 in Figure 5-6.

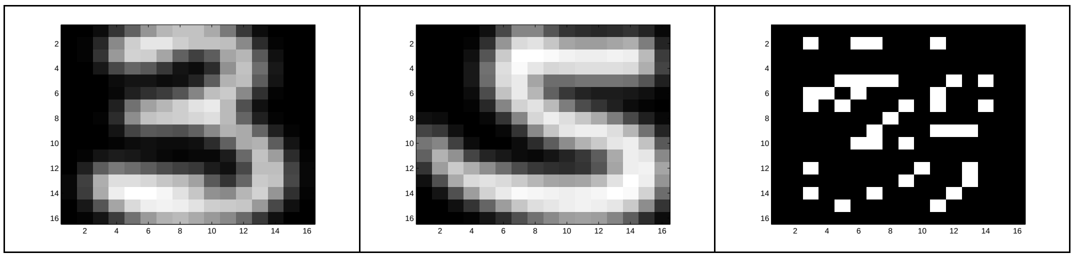


Figure 5-6: A sample of digit "3" and "5" images and the 37 significant pixels selected by ARD based SPGPC

By using fewer features, the accuracy even increases a little bit, which implies that filtering out the unrelated features can not only reduce the time complexity but also increase the accuracy.

5.4 Summary

In this chapter, inspired by the unifying framework of different sparse models [Candela and Rasmussen 2005], we toke a further step to develop the requisite calculations to extend the SPGP model to handle classification problems. Moreover, to further reduce the computation overheads caused by calculating the derivatives in high dimensional space, we proposed to use the ARD kernel [Neal 1996] in the SPGP classifier. In this way we achieve a *dual sparseness* in both the training instance space and the input feature space. And thus a dual reduction of computation costs is achieved. In fact, compared to other models, significant reductions of both training and evaluation times in SPGPC have been reported in the experiments. As a byproduct, the feature selection results generated by optimizing ARD are usually able to provide extra insights to the problems being modeled.

Chapter 6

6 Conclusions and Future Work

In the previous chapters, we have introduced the problem of learning from data in a general context, and examined the main difficulties of traditional learning models, namely their deteriorations of accuracy and explosions of computation cost when handling datasets that involve a large number of training instances and input features. We've also reviewed some successful extensions of the traditional models for regression problems. Based on these discussions, two classification models are further proposed in Chapter 3 and 5 to handle the difficulties in classification problems. In this chapter, we summarize the contributions of the thesis and discuss some possible future work.

6.1 Contributions of the Thesis

In Chapter 2 and 3, we developed the Extended Normalized Radial Basis Function Classifier (ENRBFC) based on the parametric ENRBF model [Xu 1998]. In ENRBFC, a *linear logistic* function that is similar to the linear function in ENRBF is used to model classification outputs. A training algorithm that is combined by the Expectation Maximization (EM) and the Iterative Reweighted Least Squares (IRLS) has been proposed to train the ENRBFC model. From a theoretical point of view, we analyzed why ENRBFC can improve the learning accuracy and avoid the explosion of computation time for high dimensional classification problems. By revealing the deep connection between ENRBFC and the Gaussian Mixture assumption, we also discussed some other classifiers that are related to ENRBFC. To compare ENRBFC with other models, we constructed several experiments in those chapters. From the experimental results, it is observed that ENRBFC is able to produce much better classification accuracies than the other models, especially when the number of input dimensions increases. ENRBFC is also computationally efficient because it can significantly reduce the computation time in those experiments.

In Chapter 4 and 5 we turn to explore the other main learning paradigm—nonparametric models and one of its representatives called the Gaussian Process (GP) model. We examined the main difficulties of the traditional GP model with classification problems and developed the Sparse Pseudo-input Gaussian Process Classifier (SPGPC). The SPGPC model is based on the regression model SPGP proposed by [Snelson and Ghahramani 2006; Snelson 2007] and the unifying framework of sparse models proposed by [Candela and Rasmussen 2005]. Theoretically, we developed the necessary computations that are necessary and sufficient for combining SPGP model with the Laplace approximation framework. These computations consist of the derivatives of the related matrices with respect to both kernel hyperparameters and inducing vectors. Empirically, a series of well designed classification experiments have been used to compare the proposed SPGPC with some similar models. From these experiments both the improvement of classification rates and the reduction of computation times are observed with SPGPC. These improvements are especially significant when the classification problems involve hundreds and thousands of training instances. In the SPGPC model, we also proposed to use the *automatic relevance determination* (ARD) kernel function to generate a rank of relative importance of input features. This ranking information enables us to do the feature selection by picking up the most significant features and thus effectively reduce the problems' input dimensionality. As a result, the SPGPC's vulnerability to high dimensional computations has been largely avoided. This claim has been supported by the experimental results.

6.2 Discussions on Future Work

In the future there are still a lot of areas to be explored about this work. For example, it will be interesting to compare ENRBFC and SPGPC in a systematic way. In Section 5.2 we briefly discussed the differences and connections between ENRBFC and SPGPC. So a practical question is which model is more appropriate to what kinds of classification problems. In [Neal 1996] Neal has proven that for a broad class of prior distributions over the parameters, the distributions of functions that can be represented by a parametric model (e.g. RBF) will tend to a nonparametric Gaussian Process model, in the limit of an infinite number of basis functions used by RBF. At the first glance, it seems that the performance of a GP should always be better than a RBF classifier, in view of that a GP is equivalent to using infinite number of basis functions in a RBF. However, there is evidence showing that as an infinite limit, the GP model could lose some important things compared with RBF. For example, for a classification problem that has more than one output (which is very common in practice), there are non-trivial correlations between the outputs in a RBF model, because these outputs share their inputs from the same set of basis functions. But in a GP model as the limit of an infinite number of basis functions, these correlations vanish [MacKay 1998; Bishop 2006]. As a result, a RBF model could provide more accurate modeling of the data in a classification problem. Thus, more efforts are deserved to compare the performance of the two types of classifiers in an empirical way.

Besides the comparison of learning performances, it is also interesting to find out the relations between the basis functions in ENRBFC and the inducing vectors in SPGPC. Particularly, the kernel functions centered at the inducing vectors in SPGPC can be viewed as the analogical counterparts of the basis functions in ENRBFC. So another direction for future work is to compare the basis functions and the inducing vectors in terms of their numbers, centers and corresponding shapes. These comparisons could benefit both types of models. For example, finding inducing vectors in SPGPC is relatively expensive compared to finding centers of basis functions in ENRBFC. And the ARD kernel in SPGPC could provide relative importance of different features that is lacked in ENRBFC. If a relation between the two types of parameters is found, the different models will be able to take advantages of the results from the other. Furthermore, besides GP, there are also some sparse models from other kernel methods, such as the Support Vector Machine (SVM) [Vapnik 1995] and Relevance Vector Machine (RVM) [Tipping 2001]. It will be interesting to compare the performance of these different sparse kernel methods and the relations between their corresponding sparseness parameters.

In the future the individual models proposed in the thesis can also be improved in several ways. For example, in the current version of ENRBFC, a model selection process that is based on the k-fold cross validation is used to determine the appropriate model complexity. As an alternative, we can consider using the parameter shrinkage methods such as LASSO [Tibshirani 1996] or weight decay [Bishop 1995] to *regularize* the weight parameters of basis functions. A pleasant result of these shrinkage methods is that they can provide some hints about the relative importance of different features. This way the feature selection results can also be provided by ENRBFC, just as in SPGPC.

As for the SPGPC model, our focus in the future will include simplifying the calculations involved in the current approximation and developing new methods of determining inducing vectors in a cheaper way. The SPGPC model could be further enhanced by borrowing the idea of "mixture of expert" from the ENRBFC model. With this idea we can extend the SPGPC model to the *mixtures of Gaussian process models* [Rasmussen and Williams 2006].

In detail, a mixture of different ENRBFC models will be fit in the input space, each of which models a local region of the input space. This is similar to dividing the input space by different basis functions in ENRBFC. As a result, a mixture stochastic process model can be achieved, which is more flexible than an individual local GP model. Similar to the EM algorithm for ENRBFC, The Markov Chain Monte Carlo (MCMC) [Bishop 2006] method will be used to estimate latent memberships of different input instances to the different local models. In the literature, Rasmussen and Ghahramani [Rasmussen and Ghahramani 2002] has developed a mixture of expert model by using the traditional GP as local experts and the Dirichlet process as the latent membership distributions. However, this work is originally proposed for regression problems. So its extension to classifications will be considered in the future.

On the experiment side, we consider testing the proposed models with more real-world applications that have a larger number of instances and input features such as genetic medical data or online document classifications.

Appendix

Mathematical Background

In this appendix we briefly describe the mathematical tools that are repeatedly used throughout the thesis. More details can be referred to [Rasmussen and Williams 2006] and [Magnus and Neudecker 1999].

• Conditional Probability from Joint Gaussian Distribution

Assume $\mathbf{x}$ and $\mathbf{y}$ follow a joint Gaussian distribution, i.e.,

$$p(\mathbf{x},\mathbf{y}) = \mathrm{N}\left(\begin{bmatrix}\boldsymbol{\mu}_{\mathrm{x}}\\ \boldsymbol{\mu}_{\mathrm{y}}\end{bmatrix}, \begin{pmatrix}\mathbf{K}_{\mathrm{x}} & \mathbf{K}_{\mathrm{xy}}\\ \mathbf{K}_{\mathrm{yx}} & \mathbf{K}_{\mathrm{y}}\end{pmatrix}\right) \tag{1}$$

Then the marginal and conditional probability can be represented by:

$$p(\mathbf{x}) = \mathrm{N}\left(\boldsymbol{\mu}_{\mathrm{x}}, \mathbf{K}_{\mathrm{x}}\right) \tag{2}$$

$$p(\mathbf{x}\,|\,\mathbf{y}) = \mathrm{N}\left(\boldsymbol{\mu}_{\mathrm{x}} + \mathbf{K}_{\mathrm{xy}}\mathbf{K}_{\mathrm{y}}^{-1}\left(\mathbf{y}-\boldsymbol{\mu}_{\mathrm{y}}\right), \mathbf{K}_{\mathrm{x}} - \mathbf{K}_{\mathrm{xy}}\mathbf{K}_{\mathrm{y}}^{-1}\mathbf{K}_{\mathrm{yx}}\right) \tag{3}$$

• Product of Two Gaussian Distributions

The product of a Gaussian in $\mathbf{x}$ and a Gaussian in linear form $\mathbf{Px}$ is an *unnormalized* Gaussian in $\mathbf{x}$:

$$\mathrm{N}\left(\mathbf{x}\,|\,\mathbf{a},\mathbf{A}\right)\mathrm{N}\left(\mathbf{Px}\,|\,\mathbf{b},\mathbf{B}\right) = z\mathrm{N}\left(\mathbf{x}\,|\,\mathbf{c},\mathbf{C}\right) \tag{4}$$

Where

$$\begin{aligned}\mathbf{c} &= \mathbf{C}\left(\mathbf{A}^{-1}\mathbf{a} + \mathbf{P}^{T}\mathbf{B}^{-1}\mathbf{b}\right)\\ \mathbf{C} &= \left(\mathbf{A}^{-1} + \mathbf{P}^{T}\mathbf{B}^{-1}\mathbf{P}\right)^{-1}\end{aligned} \tag{5}$$

and z is the normalizing factor,

$$z = (2\pi)^{-\frac{D}{2}} \left|\mathbf{B}+\mathbf{PAP}^T\right|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{b}-\mathbf{Pa})^T\left(\mathbf{B}+\mathbf{PAP}^T\right)^{-1}(\mathbf{b}-\mathbf{Pa})\right) \tag{6}$$

The formula of z can also be used in the calculation of the integral,

$$z = \int \mathrm{N}\left(\mathbf{x} \mid \mathbf{a}, \mathbf{A}\right) \mathrm{N}\left(\mathbf{Px} \mid \mathbf{b}, \mathbf{B}\right) d\mathbf{x} \tag{7}$$

- **Matrix Inversion Lemma**

The matrix inversion lemma (or Woodbury, Sherman & Morrison formula) is known as:

$$(\mathbf{Z}+\mathbf{UWV})^{-1} = \mathbf{Z}^{-1} - \mathbf{Z}^{-1}\mathbf{U}\left(\mathbf{W}^{-1}+\mathbf{VZ}^{-1}\mathbf{U}\right)^{-1}\mathbf{VZ}^{-1} \tag{8}$$

assuming that the relevant inverses exist. A similar representation exists for the determinant of matrices:

$$\left|\mathbf{Z}+\mathbf{UWV}\right| = \left|\mathbf{Z}\right|\left|\mathbf{W}\right|\left|\mathbf{W}^{-1}+\mathbf{VZ}^{-1}\mathbf{U}\right| \tag{9}$$

- **Matrix Differential and Kronecker Product**

$$d\mathbf{F}^{-1} = -\mathbf{F}^{-1}(d\mathbf{F})\mathbf{F}^{-1} \tag{10}$$

$$d\,\mathbf{vec}(\mathbf{F}) = \mathbf{vec}(d\mathbf{F}) \tag{11}$$

$$\mathbf{vec}(\mathbf{ABC}) = \left(\mathbf{C}^T \otimes \mathbf{A}\right)\mathbf{vec}(\mathbf{B}) \tag{12}$$

Summary of Publications

International Journals

[1] Ma. Li, A. Wahab, Ng. G. See, and S. S. Erdogan, "An experimental study of the extended NRBF regression model and its enhancement for classification problem," *Neurocomputing*, vol. 72, 2008.

[2] Ma. Li, A. Wahab, and Ng. G. See, "An enhanced application of SPGP model in classification problems," *Neurocomputing*, 2009. *Submitted.*

International Conference

[1] Ma. Li, A. Wahab, and C. Quek, "A New Algorithm for Order-Independent Clustering," presented at ISCA 18th International Conference on Computer Applications in Industry and Engineering, Honolulu, Hawaii, USA, 2005. *Nominated for the best paper award*.

[2] Ma. Li, A. Wahab, and C. Quek, "A Modified Generalized RBF Model with EM-based Learning Algorithm for Medical Applications," presented at The 19th IEEE International Symposium on Computer-Based Medical Systems, Salt Lake City, USA, 2006.

[3] Ma. Li, A. Wahab, and C. Quek, "A Modified Generalized RBF Model with EM-based Learning Algorithm," presented at International Conference on Artificial Intelligence, Monte Carlo Resort, Las Vegas, Nevada, USA, 2006.

[4] Ma. Li, T. Kaixiang, A. Wahab, and S. Tin, "Finding Useful Features in Calcium Scoring Prediction--a medical application of different feature selection methods," presented at IEEE International Workshop on Digital InfoTainment and Visualization (IWDTV06), Singapore, 2006.

[5] Ma. Li, A. Wahab, and T. Kaixiang, "Emotion Recognition Based on EEG Time Domain Feature Extraction," presented at International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2007), San Diego, USA, 2007.

Book Chapters

[1] Ma. Li, Q. Chai, T. Kaixiang, A. Wahab, and H. Abut, "Chapter 10: EEG Emotion Recognition System," in *In-Vehicle Corpus and Signal Processing for Driver Behavior*, K. Takeda, H. Erdogan, J. H. L. Hansen, and H. Abut, Eds.: Springer, 2008.

Bibliography:

Albrecht, S., J. Busch, et al. (2000). "Generalized radial basis function networks for classification and novelty detection: self-organization of optimal Bayesian decision." Neural Networks **13**: 1075-1093.

Bellman, R. E. (1961). Adaptive Control Processes, Princeton University Press.

Bishop, C. (1995). Neural Networks for Pattern Recognition, Oxford: Clarendon Press.

Bishop, C. M. (2006). Pattern Recognition and Machine Learning, Springer.

Bugmann, G. (1998). "Normalized Gaussian Radial Basis Function Networks." Neurocomputing **20**: 97-110.

Candela, J. Q. and C. E. Rasmussen (2005). "A Unifying View of Sparse Approximate Gaussian Process Regression." Journal of Machine Learning Research **6**: 1939-1959.

Cherkassky, V. and F. Mulier (1998). Learning From Data: Concepts, Theory, and Methods, John Wiley & Sons, Inc.

Csato, L. and M. Opper (2002). "Sparse Online Gaussian Processes." Neural Computation **14**(3): 641-669.

Dempster, A. P., N. M. Laird, et al. (1977). "Maximum likelihood from incomplete data via the EM algorithm (with discussion)." J. Roy. Stat. Soc. **B39**: 1-38.

Duda, R. O., P. E. Hart, et al. (2001). Pattern Classification. New York, John Wiley & Sons, Inc.

Fletcher, R. (1987). Practical Methods of Optimization (2nd), Wiley.

Gan, Q. and C. J. Harris (2001). "A hybrid learning scheme combining EM and MASMOD algorithms for fuzzy local linearization modeling." IEEE Trans. on Neural Networks **12**(1): 44-53.

Geng, T., J. Q. Gan, et al. (2008). "A novel design of 4-class BCI using two binary classifiers and parallel mental tasks." Journal of Computational Intelligence and Neuroscience **2008**.

Gibbs, M. N. and D. J. MacKay (2000). "Variational Gaussian process classifiers." IEEE Transactions on Neural Networks **11**(6).

Golub, G. H. and C. F. V. Loan (1989). Matrix Computation (2nd). Baltimore, John Hopkins University Press.

Harris, C. J., X. Hong, et al. (2002). Adaptive Modeling, Estimation and Fusion from Data: A Neurofuzzy Approach, Springer.

Hastie, T. and R. Tibshirani (1990). Generalized Additive Models. London, Chapman and Hall.

Hastie, T. and R. Tibshirani (1996). "Discriminant Adaptive Nearest Neighbor Classification." IEEE Trans. on Pattern Analysis and Machine Intelligence **18**: 607-616.

Hastie, T., R. Tibshirani, et al. (2001). The elements of statistical learning: data mining, inference, and prediction. New York, Springer.

Hu, X. and L. Xu (2004). "A comparative investigation on subspace dimension determination." Neural Networks **17**: 1051.

Jain, L. K., M. N. Murty, et al. (1999). "Data Clustering: A Review." ACM Computing Surveys **31**(3).

Jordan, M. I. and R. A. Jacobs (1994). "Hierarchical mixture of experts and the EM algorithm." Neural Computation **6**: 181-214.

Keerthi, S. and W. Chu (2006). "A Matching Pursuit approach to sparse Gaussian process regression." Advances in Neural Information Processing Systems **18**.

Konig, H. (1986). Eigenvalue Distribution of Compact Operators, Birkhauser.

Kwok, H. F., D. A. Linkens, et al. (2003). "Rule-base derivation for intensive care ventilator control using ANFIS." Artificial Intelligence in Medicine **29**: 185-201.

Lawrence, N. D., M. Seeger, et al. (2003). "Fast sparse Gaussian process methods: the informative vector machiine." Advances in Neural Information Processing Systems **15**.

Lawrence, N. D., M. Seeger, et al. (2003). "Fast sparse Gaussian process methods: The informative vector machine." Advances in Neural Information Processing Systems **15**.

Looney, C. G. (2002). "Radial basis functional link nets and fuzzy reasoning." Neurocomputing **48**: 489-509.

Ma, L., W. Abdul, et al. (2006 A). A Modified Generalized RBF Model with EM-based Learning Algorithm for Medical Applications. The 19th IEEE International Symposium on Computer-Based Medical Systems, Salt Lake City, USA.

Ma, L., W. Abdul, et al. (2006 B). A Modified Generalized RBF Model with EM-based Learning Algorithm. International Conference on Artificial Intelligence, Monte Carlo Resort, Las Vegas, Nevada, USA.

Ma, S. and C. Ji (1998). "Fast training of recurrent networks based on the EM algorithm." IEEE Trans. on Neural Networks **9**: 11-26.

Ma, S., C. Ji, et al. (1997). "An efficient EM-based training algorithm for feedforward neural networks." Neural Networks **10**: 243-256.

MacKay, D. J. (2003). Information Theory, Inference and Learning Algorithms, Cambridge University Press.

MacKay, D. J. C. (1998). Introduction to Gaussian Process. Neural Networks and Machine Learning. C. M. Bishop, Springer**:** 133-165.

Magnus, J. R. and H. Neudecker (1999). Matrix differential calculus with applications in statistics and econometrics. New York, John Wiley.

Marcelino, L., S. Ignacio, et al. (2003). "A new EM-based training algorithm for RBF networks." Neural Networks **16**(1): 69-77.

McLachlan, G. J. and D. Peel (2000). Finite Mixture Models, Wiley.

Minka, T. (2001). A family of algorithms for approximate Bayesian inference, PhD thesis, Massachusetts Institute of Technology.

Mitchell, T. M. (1997). Machine Learning (International Edition), MIT Press & McGraw-Hill Companies, Inc.

Nabney, I. (2002). NETLAB: algorithms for pattern recognitions, Springer.

Neal, R. M. (1996). "Bayesian learning for neural networks." Lecture Notes in Statistics **118**.

Neal, R. M. (1999). Regression and Classification using Gaussian Process Priors. Bayesian Statistics. J. M. Bernardo, J. O. Berger, A. P. Dawid and A. F. M. Smith, Oxford University Press. **6**.

Neal, R. M. and G. Hinton (1998). A view of the EM algorithm that justifies incremental, sparse, and other variants. Learning in Graphical Models. J. M, Dordrecht: Kluwer Academic Publishers**:** 355-368.

Nguyen, M. N. and D. Shi (2008). "Online Bayesian Ying-Yang Learning for FCMAC." Neurocomputing **in press**.

Nguyen, M. N., D. Shi, et al. (2006). "FCMAC-BYY: Fuzzy CMAC Using Bayesian Ying Yang Learning." IEEE Transactions on Systems, Man and Cybernetics, Part B **36**(5).

Nocedal, J. and S. J. Wright (1999). Numerical Optimization, Springer.

Orr, M. J. L. (1998). An EM algorithm for regularised RBF networks. International Conference on Neural Networks and Brain. Beijing, China.

Powell, M. J. D. (1987). Radial Basis Functions for Multivariable Interpolation: A Review. Algorithms for Approximation. J. C. Mason and M. G. Cox, Oxford University Press**:** 143-167.

Press, W. H., T. S. A., et al. (1992). Numerical Recipes in C: The Art of Scientific Computing (2nd), Cambridge University Press.

Rasmussen, C. E. and Z. Ghahramani (2002). Infinite Mixtures of Gaussian Process Experts. Advances in Neural Information Processing Systems 14. T. G. Diettrich, S. Becker and Z. Ghahramani.

Rasmussen, C. E. and C. K. I. Williams (2006). Gaussian Process for Machine Learning, MIT Press.

Ripley, B. (1996). Pattern Recognition and Neural Networks. Cambridge, Cambridge University Press.

Rubin, D. B. (1983). "Iteratively reweighted least squares." Encyclopedia of Statistical Sciences **4**: 272-275.

Scholkopf, B. and A. J. Smola (2002). Learning with Kernels, MIT Press.

Seeger, M., C. K. I. Williams, et al. (2003). Fast forward selection to speed up sparse Gaussian regression. Ninth International Workshop on Artificial Intelligence and Statistics, Society for Artificial Intelligence and Statistics.

Silverman, B. W. (1985). "Some aspects of the spline smoothing approach to non-parametric regression curve fitting." Journal of Roy. Stat. Soc. B **47**(1): 1-52.

Singh, A., C. Quek, et al. (2008). "DCT-Yager FNN: A novel Yager based Fuzzy Neural Network with the Discrete Clustering Technique." IEEE Transactions on Neural Networks **19**(4): 625-644.

Smola, A. J. and P. L. Bartlett (2001). "Sparse greedy Gaussian process regression." Advances in Neural Information Processing Systems **13**.

Snelson, E. (2007). Flexible and efficient Gaussian process models for machine learning, PhD thesis. Gatsby Computational Neuroscience Unit, University College London.

Snelson, E. and Z. Ghahramani (2006). "Sparse Gaussian processes using pseudo-inputs." Advances in Neural Information Processing Systems **18**.

Staiano, A., R. Tagliaferri, et al. (2006). "Improving RBF networks performance in the regression tasks by means of a supervised fuzzy clustering." Neurocomputing **69**(13-15): 1570-1581.

Sugeno, M. and G. T. Kang (1988). "Structure identification of fuzzy model." Fuzzy Sets and Systems **28**: 15-33.

Takagi, T. and M. Sugeno (1985). "Fuzzy identification of systems and its applicaitons to modeling and control." IEEE Trans. on Systems, Man, and Cybernetics **15**.

Takahashi, K. (2000). Remarks on Emotion Recognition from Bio-Potential Signals. 2nd International Conference on Automomous Robots and Agents.

Teddy, S. D., E. M.-K. Lai, et al. (2008). "A Cerebellar Associative Memory Approach to Option Pricing and Arbitrage Trading." Neurocomputing **71**: 3303-3315.

Teddy, S. D., C. Quek, et al. (2008). "PSECMAC: A Novel Self-Organizing MultiResolution Associative Memory Architecture." IEEE Transactions on Neural Networks **19**.

Tibshirani, R. (1996). "Regression shrinkage and selection via the lasso." J. Royal. Statist. Soc. B. **58**: 267-288.

Tibshirani, R. (1996). "Regression shrinkage and selection via the lasso." Journal of the Royal Statistical Society **B**(58): 267-288.

Tipping, M. E. (2001). "Sparse Bayesian learning and the relevance vector machine." Journal of Machine Learning Research **1**: 211-244.

Vapnik, V. N. (1995). The Nature of Statistical Learning Theory. New York, Springer Verlag.

Wahba, G. (1990). Spline Models for Observational Data. CBMS-NSF Regional Conference series in applied mathematics. Philadelphia, PA, Society for Industrial and Applied Mathematics.

Wahba, G., X. Lin, et al. (1999). "The bias-variance tradeoff and the randomized GACV." Advances in Neural Information Processing Systems **11**.

Wang, C. S., D. Shaw, et al. (1998). "Intelligent control system for ventilators." Medical Engineering & Physics **20**(7): 534-542.

Williams, C. K. I. and C. E. Rasmussen (1996). "Gaussian process for regression." Advances in Neural Information Processing Systems **8**.

Xu, L. (1998). "RBF nets, mixture experts, and Bayesian Ying-Yang learning." Neurocomputing **19**: 223-257.

Xu, L. (2004). "Advances on BYY Harmony Learning: Information Theoretic Perspective, Generalized Projection Geometry, and Independent Factor Autodetermination." IEEE Trans. on Neural Networks **15**: 885-902.

Xu, L., M. I. Jordan, et al. (1995). An Alternative Model for Mixtures of Experts. Advances in Neural Information Processing Systems, MIT Press, Cambridge MA.

Zhou, S.-M., J. Q. Gan, et al. (2008). "Classifying mental tasks based on features of higher-order statistics from EEG signals in brain-computer interface." International Journal of Information Sciences **178**(6): 1629-1640.