BAYESIAN LEARNING FRAMEWORK WITH KERNEL-IMBEDDED GAUSSIAN PROCESSES APPLIED TO MICROARRAY ANALYSIS

A DISSERTATION SUBMITTED TO THE GRADUATE DEVISION OF THE UNIVERSITY OF HAWAI'I IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

COMPUTER SCIENCE

AUGUST 2008

By

Xin Zhao

Dissertation Committee:

Guylaine Poisson, Chairperson
   Stephen Itoga
   Henri Casanova
   Kyungim Baek
   Gernot Presting
We certify that we have read this dissertation and that, in our opinion, it is satisfactory in scope and quality as a dissertation for the degree of Doctor of Philosophy in Computer Science.

DISSERTATION COMMITTEE

[Signatures]
Chairperson

[Signatures]

[Signatures]

[Signatures]
Acknowledgement

Above all, I would love to present this dissertation to my parents.

I would like to dedicate this work to the memory of Dr. Will Gersch, whose teaching and help I will remember forever. I would like to express my most sincere gratitude and respect to my supervisor, Dr. Guylaine Poisson. Without her support, encouragement and intellectual advice, I would never be able to finish this work. I also would like to thank all the members in my dissertation committee, Dr. Stephen Itoga, Dr. Henri Casanova, Dr. Kyungim Baek and Dr. Gernot Presting, for their positive suggestions and understandings during the period of this work. I have greatly benefited from doing research works for Dr. Anthony Kuh, Dr. Pao-Shin Chu and Dr. Leo Wang-Kit Cheung during my study. Especially, Dr. Leo Wang-Kit Cheung initialized this project. I would like to express my appreciation for them as well.
Abstract

DNA microarray technology has provided researchers a high-throughput means to simultaneously measure expression levels for thousands of genes in an experiment. With a probit regression setting and assuming that the link function between significant gene expression data and latent variable for the response label is a Gaussian process, a kernel-induced hierarchical Bayesian framework is built for a cancer classification problem by using microarray gene expression data.

Targeting a multi-classification problem and adopting a variable selection approach with a Gibbs sample as core, we developed the algorithm, kernel-imbedded Gaussian Process (KIGP), to analyze microarray data under a Bayesian framework. Through a feature projection procedure and using a univariate ranking scheme as gene-selection strategy, we further designed an alternative microarray analysis model, natural kernel-imbedded Gaussian Process (NKIGP). In the end, embedded with a reversible jump Markov chain Monte Carlo (RJMCMC) model, we present an efficient algorithm with a cascading structure to unify the proposed methods of this study.

The simulated examples demonstrate that, our method performs almost always close to the Bayesian bound in both the cases with linear Bayesian classifiers and the cases with very non-linear Bayesian classifiers. Even with mislabeled training samples, our method is still robust, showing its broad usability to those microarray analysis problems that linear methods may work flakily.
Six published microarray datasets were analyzed in this study. The results show that predictive performance of our method for all these datasets is better than or at least as good as that of other state-of-the-art microarray analysis methods. Our method especially shows its superiority in analyzing one dataset that contains multiple suspicious mislabeled samples. For each of these datasets, we identified a set of significant genes, which can be used for further biological inspection at genome level.

In summary, built on a Gaussian process model, a kernel-induced hierarchical Bayesian framework using microarray gene expression data for a cancer multi-classification problem is presented in this study. Our main contribution is a fully automated learning algorithm to solve this Bayesian model. Satisfactory results have been achieved in both the simulated examples and the real-world data studies.
# Table of Contents

Acknowledgements ................................................................................................................. iv

Abstract................................................................................................................................... v

Table of Contents ....................................................................................................................... vii

List of Tables .............................................................................................................................. x

List of Figures ............................................................................................................................ xii

List of Abbreviations .................................................................................................................. xiv

1 Background Introduction and Problem Statement ................................................................. 1

1.1 Microarray Analysis and Alternative Approach Review ................................................... 1

1.2 Problem Formulation and Gaussian Process .................................................................... 8

1.2.1 General Model Formulation ......................................................................................... 8

1.2.2 Feature Space and Kernel-Induced Learning Methods ................................................. 12

1.2.3 Markov Chain Monte Carlo Method ............................................................................ 15

1.2.4 Gaussian Processes for Microarray Analysis ............................................................... 17

1.2.5 Illustrative Examples ................................................................................................... 19

1.3 General Framework of this Study ..................................................................................... 27

2 KIGP Framework for Microarray Analysis ......................................................................... 30

2.1 KIGP Framework for a Binary Classification Problem .................................................... 32

2.1.1 Prior Specifications .................................................................................................... 33
2.1.2 Derivation of the Computational Implementation of a Gibbs Sampler
2.1.3 Kernel Parameter Tuning
2.1.4 Proposed Gibbs Sample
2.1.5 Overall Algorithm
   2.1.5.1 Kernel Parameter Fitting Phase
   2.1.5.2 Gene Selection Phase
   2.1.5.3 Prediction Phase
   2.1.5.3 Discussion on the Simplified Procedures
2.2 KIGP Framework for a Multi-Classification Problem
   2.2.1 Derivation of the Computational Implementation of a Gibbs Sampler
   2.2.2 Proposed Gibbs Sample
   2.2.3 Prediction
2.3 Simulation Results and Discussions
   2.3.1 Predictive Fit Measure Approaches to Model Performance Assessment
   2.3.2 Simulated Examples
      2.3.2.1 Examples with a Binary Classification Model
      2.3.2.2 Examples with a Multi-Classification Model
      2.3.2.3 An Example with a Mislabeled Training Sample
   2.3.3 Real Dataset Studies
      2.3.3.1 Binary Datasets
      2.3.3.2 Multi-Class Datasets
      2.3.3.3 Discussion on the Generalized Performance
3 Building Kernels from Microarray: a Natural Kernel Approach .............................................. 94
  3.1 Basic Concepts of Natural Kernel ......................................................................................... 95
  3.2 Natural Kernel Applied to a Microarray Classification Problem ...................................... 100
    3.2.1 Natural Gaussian Fisher Kernel (NGFK) ................................................................. 101
    3.2.2 Natural Student-t Fisher Kernel (NTFK) ................................................................. 107
    3.2.3 Issues on Implementation ......................................................................................... 111
  3.3 Simulation Results and Discussions .................................................................................. 114
    3.3.1 Simulated Examples ................................................................................................. 115
    3.3.2 Real Data Studies ...................................................................................................... 123

4 Simultaneous Kernel Type Competition: a RJMCMC Approach .......................................... 128
  4.1 Introduction to the General RJMCMC algorithm ............................................................... 131
  4.2 RJMCMC Designed for the Kernel Type Competition Problem .................................... 134
    4.2.1 Scenario 1: RJMCMC for Prediction Phase ................................................................ 135
    4.2.2 Scenario 2: RJMCMC for Pre-Prediction Phase ......................................................... 144
  4.3 Conclusive Algorithm for KIGP Microarray Analysis and Simulation Results ........... 157

5 Summary and Discussions ........................................................................................................ 162

Annex: Supplementary Documents ............................................................................................ 167
  A. Descriptions of the Selected Genes for the Real Datasets .................................................. 167
  B. Heatmaps of the Selected Genes for the Real Datasets ...................................................... 169
  C. Link to C++ Code and Example Datasets ........................................................................... 172

Bibliography ................................................................................................................................ 173
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Performance comparison of different methods for the simulated multi-classification examples</td>
</tr>
<tr>
<td>2-2</td>
<td>Summary of the datasets analyzed in this study</td>
</tr>
<tr>
<td>2-3</td>
<td>Performance results for the binary real datasets</td>
</tr>
<tr>
<td>2-4</td>
<td>Index of the significant genes selected by KIGP for the real binary datasets</td>
</tr>
<tr>
<td>2-5</td>
<td>Performance results of KIGP for the real multi-class datasets</td>
</tr>
<tr>
<td>2-6</td>
<td>Index of the significant genes selected by KIGP for the real multi-class datasets</td>
</tr>
<tr>
<td>2-7</td>
<td>Performance comparison of different methods for the real multi-class datasets</td>
</tr>
<tr>
<td>2-8</td>
<td>Performance comparison between the 3-fold CV procedure and the independent testing for the leukemia dataset and the SRBCT dataset</td>
</tr>
<tr>
<td>2-9</td>
<td>Performance comparison by applying different classifiers to the colon dataset</td>
</tr>
<tr>
<td>3-1</td>
<td>Performance comparison Example 1 (3.3.1)</td>
</tr>
<tr>
<td>3-2</td>
<td>Testing results for the real datasets by using NKIGP</td>
</tr>
<tr>
<td>3-3</td>
<td>Performance comparison for different classifiers applied to the colon dataset</td>
</tr>
<tr>
<td>4-1</td>
<td>Estimated posterior probabilities for the simulated examples (4.3) using KIGP_RJ algorithm</td>
</tr>
<tr>
<td>4-2</td>
<td>Estimated posterior probabilities for the real datasets using KIGP_RJ algorithm</td>
</tr>
</tbody>
</table>
A-1  Gene description of the genes selected by KIGP/LK for the leukemia dataset……167
A-2  Gene description of the genes selected by KIGP/PK for the SRBCT dataset……168
A-3  Gene description of the genes selected by KIGP/GK for the colon dataset………168
A-4  Gene description of the genes selected by KIGP/GK for the breast cancer dataset.................................................................................................................................169
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Diagram of a typical DNA microarray experiment</td>
</tr>
<tr>
<td>1-2</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Illustration of the feature mapping and feature space concept</td>
</tr>
<tr>
<td>1-3</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Reconstructed functions of Example 1 (1.2.5)</td>
</tr>
<tr>
<td>1-4</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>MSE plot of Example 1 (1.2.5)</td>
</tr>
<tr>
<td>1-5</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>Class decision function plot of Example 2 (1.2.5)</td>
</tr>
<tr>
<td>1-6</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>MR plot of Example 2 (1.2.5)</td>
</tr>
<tr>
<td>1-7</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Class decision function plot of Example 3 (1.2.5)</td>
</tr>
<tr>
<td>1-8</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>MR comparison plot of Example 2 and 3 (1.2.5)</td>
</tr>
<tr>
<td>1-9</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Flow chart for the general framework of this study</td>
</tr>
<tr>
<td>2-1</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Directed acyclic graph for Bayesian analysis of a KIGP model</td>
</tr>
<tr>
<td>2-2</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>Interim results of the KIGP/GK simulations for Example 1 and 2 (2.3.2.1)</td>
</tr>
<tr>
<td>2-3</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>Results of the KIGP simulations for Example 1 (2.3.2.1)</td>
</tr>
<tr>
<td>2-4</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>Results of the KIGP simulations for Example 2 (2.3.2.1)</td>
</tr>
<tr>
<td>2-5</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>Results of applying KIGP to one of training sets in example 3 (2.3.2.1)</td>
</tr>
<tr>
<td>2-6</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>Results of the KIGP simulations for Example 1 (2.3.2.2)</td>
</tr>
<tr>
<td>2-7</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>Contours of the posterior predictive probabilities of Example 1 (2.3.2.2)</td>
</tr>
<tr>
<td>2-8</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>Results of the KIGP simulations for Example 2 (2.3.2.2)</td>
</tr>
</tbody>
</table>
2-9 Contours of the posterior predictive probabilities of Example 2 (2.3.2.2) ..........70
2-10 Results of the KIGP simulation (2.3.2.3) ..................................................73
2-11 Contours of the posterior predictive probabilities (2.3.2.3) .......................74
2-12 NLF plots for the real binary datasets ...............................................................79
2-13 Estimated posterior PDFs of the width of GK for the real binary datasets.........80
2-14 Interim result plots for the KIGP simulations for the real multi-class datasets.....86
3-1 Class decision function plots for the first case in Example 1 (3.3.1) ...............116
3-2 Class decision function plots for the second case in Example 1 (3.3.1) ..........118
3-3 Class decision function plots for Example 2 (3.3.1) .......................................120
3-4 Intermediate results of the gene selection phase of Example 2 (3.3.1) ..........121
3-5 MR plot by applying NKIGP with different number of significant genes to the colon
dataset ......................................................................................................................127
4-1 Flow chart of the KIGP_RJ algorithm ...............................................................156
A-1 Heatmap of the identified genes by KIGP/LK for the leukemia dataset ..........169
A-2 Heatmap of the identified genes by KIGP/PK for the SRBCT dataset .............170
A-3 Heatmap of the identified genes by KIGP/GK for the colon dataset ................170
A-4 Heatmap of the identified genes by KIGP/GK for the lymphoma dataset .........171
A-5 Heatmap of the identified genes by KIGP/GK for the breast cancer dataset ....171
A-6 Heatmap of the identified genes by KIGP/LK for the brain tumor dataset .......172
<table>
<thead>
<tr>
<th>Full Name/Term</th>
<th>Abbreviations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acute lymphoblastic leukemia</td>
<td>ALL</td>
</tr>
<tr>
<td>Acute myeloid leukemia</td>
<td>AML</td>
</tr>
<tr>
<td>Akaike information criterion</td>
<td>AIC</td>
</tr>
<tr>
<td>Automatic relevance determination</td>
<td>ARD</td>
</tr>
<tr>
<td>Average predictive probability</td>
<td>APP</td>
</tr>
<tr>
<td>Artificial neural network</td>
<td>ANN</td>
</tr>
<tr>
<td>Bayesian information criterion</td>
<td>BIC</td>
</tr>
<tr>
<td>Breast cancer</td>
<td>BRCA</td>
</tr>
<tr>
<td>Cholesky decomposition</td>
<td>CD</td>
</tr>
<tr>
<td>Complementary DNA</td>
<td>cDNA</td>
</tr>
<tr>
<td>Cross-validation</td>
<td>CV</td>
</tr>
<tr>
<td>Desoxyribonucleic acid</td>
<td>DNA</td>
</tr>
<tr>
<td>Diagonal linear discriminant analysis</td>
<td>DLDA</td>
</tr>
<tr>
<td>Ewing family of tumors</td>
<td>EMS</td>
</tr>
<tr>
<td>Expectation-maximization</td>
<td>EM</td>
</tr>
<tr>
<td>Exponential kernel</td>
<td>EK</td>
</tr>
<tr>
<td>False discovery rate</td>
<td>fdr</td>
</tr>
<tr>
<td>Term</td>
<td>Abbreviation</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Independent and identically distributed</td>
<td>IID</td>
</tr>
<tr>
<td>Kernel Fisher discrimination</td>
<td>KFD</td>
</tr>
<tr>
<td>Kernel-imbedded Gaussian process</td>
<td>KIGP</td>
</tr>
<tr>
<td>K nearest neighbor</td>
<td>kNN</td>
</tr>
<tr>
<td>Gaussian kernel</td>
<td>GK</td>
</tr>
<tr>
<td>Gaussian process</td>
<td>GP</td>
</tr>
<tr>
<td>Gaussian process embedded with ARD parameter</td>
<td>GP_ARD</td>
</tr>
<tr>
<td>Generalized comparative Kullback-Leibler</td>
<td>GCKL</td>
</tr>
<tr>
<td>Hidden Markov model</td>
<td>HMM</td>
</tr>
<tr>
<td>Inverted Gamma</td>
<td>IG</td>
</tr>
<tr>
<td>Least angle regression</td>
<td>LAR</td>
</tr>
<tr>
<td>Least square support vector machine</td>
<td>LSSVM</td>
</tr>
<tr>
<td>Leave-one-out cross-validation</td>
<td>LOOCV</td>
</tr>
<tr>
<td>Linear kernel</td>
<td>LK</td>
</tr>
<tr>
<td>Manhattan kernel</td>
<td>MK</td>
</tr>
<tr>
<td>Markov chain Monte Carlo</td>
<td>MCMC</td>
</tr>
<tr>
<td>Mean square error</td>
<td>MSE</td>
</tr>
<tr>
<td>Messenger RNA</td>
<td>mRNA</td>
</tr>
<tr>
<td>Misclassification rate</td>
<td>MR</td>
</tr>
<tr>
<td>Multivariate normal/Gaussian</td>
<td>MN</td>
</tr>
<tr>
<td>Natural Fisher kernel</td>
<td>NFK</td>
</tr>
<tr>
<td>Natural Gaussian Fisher kernel</td>
<td>NGFK</td>
</tr>
<tr>
<td>Natural Gaussian kernel</td>
<td>NGK</td>
</tr>
<tr>
<td>Term</td>
<td>Abbreviation</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Natural kernel</td>
<td>NK</td>
</tr>
<tr>
<td>Natural kernel-imbedded Gaussian process</td>
<td>NKIGP</td>
</tr>
<tr>
<td>Natural plain kernel</td>
<td>NPK</td>
</tr>
<tr>
<td>Natural Student-t Fisher kernel</td>
<td>NTFK</td>
</tr>
<tr>
<td>Natural tangent of posterior odds kernel</td>
<td>NTOPK</td>
</tr>
<tr>
<td>Neural network</td>
<td>NN</td>
</tr>
<tr>
<td>Neuroblastoma</td>
<td>NB</td>
</tr>
<tr>
<td>Non-Hodgkin lymphoma</td>
<td>NHL</td>
</tr>
<tr>
<td>Normalized log-frequency</td>
<td>NLF</td>
</tr>
<tr>
<td>Not applicable</td>
<td>NA</td>
</tr>
<tr>
<td>Particle swarm optimization</td>
<td>PSO</td>
</tr>
<tr>
<td>Penalized logistic regression</td>
<td>PLR</td>
</tr>
<tr>
<td>Polynomial kernel</td>
<td>PK</td>
</tr>
<tr>
<td>Prediction analysis for microarray</td>
<td>PAM</td>
</tr>
<tr>
<td>Principle component</td>
<td>PC</td>
</tr>
<tr>
<td>Probability density function</td>
<td>PDF</td>
</tr>
<tr>
<td>Probability mass function</td>
<td>PMF</td>
</tr>
<tr>
<td>Random forest</td>
<td>RF</td>
</tr>
<tr>
<td>Recursive feature elimination</td>
<td>RFE</td>
</tr>
<tr>
<td>Regular kernel</td>
<td>RK</td>
</tr>
<tr>
<td>Reversible jump Markov chain Monte Carlo</td>
<td>RJMCMC</td>
</tr>
<tr>
<td>Rhabdomyosarcoma</td>
<td>RMS</td>
</tr>
<tr>
<td>Ribonucleic acid</td>
<td>RNA</td>
</tr>
<tr>
<td>Term</td>
<td>Abbreviation</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Semisupervised ellipsoid ARTMAP</td>
<td>ssEAM</td>
</tr>
<tr>
<td>Sequential Monte Carlo</td>
<td>SMC</td>
</tr>
<tr>
<td>Small round blue cell tumor</td>
<td>SRBCT</td>
</tr>
<tr>
<td>Supervised principal component analysis</td>
<td>SPCA</td>
</tr>
<tr>
<td>Support vector machine</td>
<td>SVM</td>
</tr>
<tr>
<td>Tangent of posterior odds</td>
<td>TOP</td>
</tr>
<tr>
<td>Univariate ranking</td>
<td>UR</td>
</tr>
<tr>
<td>Winning regular kernel</td>
<td>WRK</td>
</tr>
</tbody>
</table>
Chapter 1

Background Introduction and
Problem Statement

1.1 Microarray Analysis and Review of Previous Work

The basic unit of all living organisms is the cell. All the hereditary information of a cell is stored on the chromosomes that are made up from desoxyribonucleic acid, also known as DNA. DNA codes hereditary information in a double-stranded helix. A gene corresponds to a specific DNA fragment and can be interpreted as a construction manual for a protein.

Generally speaking, all cells in the same organism have the same genome, but some of the genes can be expressed differently at different times and under different conditions. The different patterns of gene expression following carefully tuned biological programs/mechanisms, according to tissue type, developmental stage, environment and genetic background, account for the enormous variety of different cell types. According to the fundamental theory of molecular biology, the way from gene to protein consists of two steps. First gene is transcribed to mRNA; then mRNA is translated into protein. Therefore, all major differences in cell state or type are virtually related to changes in the mRNA level and the genome level.

Microarray technology has provided researchers a high-throughput means to simultaneously measure expression levels for thousands of genes in an experiment. An
incomplete list of some prior outstanding research work using microarray technology includes [8], [11], [12], [23], [24] and [62]. It turns out that, careful analyses of microarray gene expression data can help better understand human health and disease and have very important implications in the basic sciences as well as in pharmaceutical and clinical research.

![Diagram of a typical DNA microarray experiment](image)

**Fig. 1-1: Diagram of a typical DNA microarray experiment ([74])**

In Fig. 1-1, we show the procedure of a typical DNA microarray experiment to get a microarray of gene expression data. In detail, to initialize the experiment, two samples are needed. One is from normal cells as reference and the other is from cancer cells for test. To construct a microarray, each investigated gene is assigned to a specific spot on the microarray. In the first step, mRNAs are extracted from the two samples. In
the second step, the mRNA molecules are reversed transcribing into the cDNAs and they are labeled with fluorescent dyes, red to cDNAs from the cancer sample and green to cDNAs from the normal sample. In the third step, the labeled cDNA mixture solutions are applied to the microarray, where the competitive binding will be done. In the end, via the use of a scanner and a computer, the ratio of the intensities of the red to green signals at each spot is obtained through image analysis.

**Microarray technology opens the possibility of creating data sets of molecular information to represent many systems of biological or clinic interest.** Gene expression profiles can be used as input, for instance, to serve as fingerprints to build more accurate molecular classification of particular disease outcome or subtype. Some earlier microarray analysis studies, such as [23], [24], [58] and [68], demonstrated that gene expression data can be used in a variety of class discovery or class prediction in biomedical problems, including those that are relevant to tumor classification. Owing to the high cost associated with microarray technology, a typical microarray study may only be done with a small number of samples (often less than one hundred), while the number of investigated genes usually is very large (typically in thousands). This nature of microarray data analysis gives rise to a very high likelihood of finding a lot of “false positives” only by chance. This represents the core challenges in microarray data analysis. A comprehensive review on microarray research can be found in [54]. In this study, we shall mainly focus on the problem of classifying cancer type with given microarray gene expression datasets. More specifically, we shall limit the scope of this study to a supervised learning problem. We assume there is a set of training samples available. Each sample contains its microarray gene expression data and its corresponding tumor type.
A typical microarray contains the expression data of thousands of genes, most of which are not relevant to the tumor types of interest. Therefore, a microarray analysis problem almost always contains two phases. In the first phase, a set of genes are identified or selected as being significantly important for the target problem. This is often termed the gene selection problem. With this set of genes, in the second phase, a prediction for the tumor type of interest is made. In this study, we shall focus on both phases to develop a microarray analysis framework, towards an optimum cancer classification problem.

The gene selection problem usually is referred as the variable selection problem in a general large-scale data analysis problem. In essence, a variable selection problem arises when one wants to model the relationship between a covariate of interest and a set of potentially important explanatory variables or predictors, but there is uncertainty on which subset should be used. The implication of the variable selection problem is that, by effectively identifying the subset of important predictors, it will improve estimation and prediction accuracy and enhance model interpretability.

Most classical variable selection methods, such as $C_p$ ([46]), Akaike Information Criteria (AIC, [2]), Bayesian Information Criteria (BIC, [66]), Least Angle Regression (LAR, [18]), are not appropriate for a typical microarray analysis problem because their relative model selection criterion has to be evaluated for each candidate model in order to select the best one. Since the number of candidate models grows at an exponential rate with an increasing number of overall investigated variables, whereas the typical number of predictors in a microarray analysis problem is in thousands, the computation
complexity is prohibitive in practice. Due to this challenge, several other approaches have been proposed to solve the gene selection problem.

Two groups of methods have been extensively studied for the gene selection problem: *filter methods* and *wrapper methods* ([77]). The essential difference between these two approaches is that a wrapper method makes use of the algorithm that will be used to build the final learner, whereas a filter method does not. Thus, given a set of variables, a wrapper method is designed to associate with the performance of the provided learner on each tested subset. A filter method, on the other hand, attempts to find predictive subsets of the variables by making use of simple statistics computed from an empirical distribution.

Generally, a filter method is relatively easier to implement than a wrapper method. For example, [24] introduced a simple univariate ranking (UR) criterion to select significant genes for binary classification problem. In [15], the authors discussed different approaches to applying a multiple hypothesis testing to microarray while controlling a suitably defined error rate. [17] developed an empirical Bayes scheme to estimate the null hypothesis of a multiple hypothesis testing by using a local version of the false discovery rate to examine the Bayesian inference issues. In [17], the author applied a multiple t-test to two microarray datasets and showed that his method is effective to find the significant genes.

Comparatively, wrapper methods require more computation time, but they usually perform better than filter methods. Many wrapper methods have been intensively studied. For example, under a generalized linear regression model and by introducing a technique called “Supervised Principal Component” analysis (SPCA), [5] presented an effective
way to select the significant subset of predictors for a supervised learning problem with large-scale data. Nevertheless, choosing the threshold for predictor selection in a SPCA is still fairly subjective and empirical. Based on the simple nearest centroid classifier and via a prototype shrinking strategy, [69] developed the “nearest shrunken centroids”, also known as “Predication Analysis of Microarray” (PAM), as one of the benchmark algorithms for microarray analysis. [26] introduced the Recursive Feature Elimination (RFE) algorithm, through which one can utilize a support vector machine (SVM) to select significant genes for a cancer classification problem via using microarray. However, it turns out that RFE does not necessarily give the optimum solution for a parametric kernel ([79]). By adopting RFE and univariate ranking ([16]) as the gene selection strategies, [85] used penalized logistic regression (PLR) to analyze microarrays, an efficient implementation of which is further provided in [84]. By designing an iterative gene set elimination strategy, [14] embedded random forest algorithm into a microarray analysis model. More recently, [78] suggested an algorithm named Semisupervised Ellipsoid ARTMAP (ssEAM) to analyze a microarray and a discrete version of Particle Swarm Optimization (PSO) was employed to indicate whether or not a gene should be chosen. Each of these methods has been proven to be successful in analyses of microarray data.

Another viable approach is to build a model under a hierarchical Bayesian framework and analyze it via Bayesian inference. There are two major ways to set up a hierarchical probabilistic model. One is through a generative model for the microarray and the other is to adopt a discriminative method. With a generative approach, for example, by assuming a hierarchical mixture Gaussian distribution to differential gene expression data, [43] used a Bayesian model to select significant genes. With a
discriminative approach, on the other hand, one usually needs to suggest a class decision function defined over the microarray. Most previous works have focused on linear (or generalized linear) methods. Via introducing an automatic relevance determination (ARD) parameter for each gene while applying a prior favoring sparse hypothesis, [39] proposed a Bayesian microarray analysis method that is embedded with the gene selection function. By embedding ARD parameter into the covariance matrix of a Gaussian Process and adopting an ordeal regression model, [9] developed another gene selection algorithm for microarray analysis. Based on a linear probit regression setting, [38] suggested a Bayesian hierarchical model and designed a Gibbs sampler to solve it. An extension to a multi-classification problem based on a multinomial probit regression model was discussed in [83]. Built on a logistic regression setting, [80, 82] applied a Bayesian approach to the gene selection problem for cancer classification by using microarray profiling. The authors also designed a Gibbs sampler to solve the Bayesian model. All these methods have been shown to achieve various levels of effectiveness in finding the significant set of genes in a wide range of real experiments. However, these linear models all share three limitations: first, a linear model is not necessarily always a good approximation for the underlying biological model; second, it has been argued that linear methods might be more sensitive to outlier samples ([55]); third, the computations of these linear model based algorithms usually involve calculating inverse of a matrix that may be singular when the number of the selected significant genes is relatively large (such as in [38]). To overcome these disadvantages, for example, [81] introduced a nonlinear term into the linear probit regression model and applied a bootstrap procedure
to enlarge the sample size. A technique called sequential Monte Carlo (SMC) was used in the numerical Bayesian computation.

1.2 Problem Formulation and Gaussian Process

1.2.1 General Model Formulation

In this study, we shall focus on analyzing differential gene expression data for a cancer type classification problem. We shall introduce a general hierarchical Bayesian model, which works as the basis of our proposed method.

Throughout this study, we assume that there are \( n \) training samples. In each sample, there are \( p \) investigated genes (usually \( p \gg n \)) and a cancer type label that is assumed to be dependent on the gene expression data. We denote the label of training samples by \( y = [y_1, y_2, ..., y_n]' \). For a binary classification problem, by convention, we denote the label by \( y_i \in \{-1, 1\} \) \( i = 1, 2, ..., n \). For a multi-class classification problem with \( M \) target classes, we denote the label by \( y_i \in \{1, 2, ..., M\} \), \( i = 1, 2, ..., n \), where "\( M \)" represents the base class. For the microarray data, we define the training gene data matrix \( \mathbf{X} \) as:

\[
\mathbf{X} = \begin{bmatrix}
gene 1 & gene 2 & \ldots & gene p \\
X_{11} & X_{12} & \ldots & X_{1p} \\
\vdots & \vdots & \ddots & \vdots \\
X_{n1} & X_{n2} & \ldots & X_{np}
\end{bmatrix} = \begin{bmatrix}
\mathbf{x}_1' \\
\vdots \\
\mathbf{x}_n'
\end{bmatrix}
\tag{1-1}
\]

In order to avoid the possible scaling unbalance problem in \( \mathbf{X} \), before doing analysis, each of the samples usually needs to be normalized, such that its observed mean and
standard deviation are 0 and 1 respectively. This procedure is termed "normalization" or "standardization".

In order to set up a classification model, we need to introduce a latent variable and adopt a regression model. We assume the latent response variable is denoted by \( z = [z_1, z_2, \ldots, z_n]^\top \). Then we define the link between the latent variable \( z \) and the data matrix \( X \) by

\[
z = g'(X) + e \tag{1-2}
\]

where, symbol \( g'(\bullet) \) denotes the link function between \( X \) and \( z \); \( e \) is the additive noise.

For a gene selection problem, a so-called "gene-selection vector" \( \gamma \) is further defined as

\[
\gamma = [\gamma_1, \gamma_2, \ldots, \gamma_p]^\top
\]

where \( \gamma_j = \begin{cases} 1 & \text{if the } j\text{th gene is selected} \\ 0 & \text{otherwise} \end{cases}, \ j = 1, 2, \ldots, p \tag{1-3} \)

The significant gene data matrix corresponding to the given gene-selection vector \( \gamma \), \( X_\gamma \), is formulated by:

\[
X_\gamma = \begin{bmatrix}
X_{11}, X_{12}, \ldots, X_{1q} \\
\vdots \\
X_{n1}, X_{n2}, \ldots, X_{nq}
\end{bmatrix} = \begin{bmatrix}
x_{\gamma_1} \\
\vdots \\
x_{\gamma_p}
\end{bmatrix} \tag{1-4}
\]

where the \( j \)th column of \( X_\gamma \) is the \( i \)th column of the matrix \( X \) while the index of the \( j \)th non-zero element in the vector \( \gamma \) is \( i \). In equation (1-4), there are \( q \) genes selected out from the total \( p \) genes and generally \( q \ll p \). If we assume that the response variable is only depended on the significant genes and the function \( g'(\bullet) \) in model (1-2) is the sum of a homogeneous function \( g(\bullet) \) and an intercept term \( b \), combining with the gene-selection setting in (1-3) and (1-4), the model (1-2) becomes
In this study, we adopt a probit regression model to build the classification model. For example, with a binary classification problem, it is

\[ z_i = g(X_{x,i}) + b + e_i \]

such that,

\[ y_i = \begin{cases} 
1 & \text{if } z_i \geq 0 \\
-1 & \text{if } z_i < 0 
\end{cases}, \quad \text{where } e_i \sim N(0, \sigma^2), \quad i = 1, 2, \ldots, n \]  

(1-6)

In a standard probit regression model, \( \sigma^2 \) is fixed as 1. The model for a multi-classification problem will be given in Section 2.2. Actually, some other regression model can also be applicable to build a classification model. For example, with a logistic regression setting, the latent variable \( z = [z_1, z_2, \ldots, z_n] \) can be instead defined for a binary-classification problem by:

\[ z_i = g(X_{x,i}) + b = \log \frac{P(y_i = 1 | X)}{P(y_i = -1 | X)}, \quad i = 1, 2, \ldots, n \]  

(1-7)

In order to improve the performance of using the model (1-6), essentially it needs to properly identify the true underlying significant genes and retrieve the best homogeneous link function \( g(\bullet) \). In principle, a unifying framework for both linear models and nonlinear models, in which model parameters can be tuned up in a convenient way, is always preferred in real applications. The kernel-based learning method is one of the approaches that show a promising potential to achieve this goal ([63, 72]).

Basically, Bayesian probability theory can help construct a unifying framework for modeling data and facilitate tuning of the involved parameter and/or hyperparameter ([19]). More importantly, Bayesian analysis provides an estimate of uncertainty in
prediction, which is an important desideratum in many real-world problems. Therefore, it is usually beneficial to develop a proper Bayesian probabilistic model for a given machine learning method. In a related work, [45] introduced the “evidence framework” as a Bayesian learning paradigm for neural networks (NNs). With the close relationship between neural network methods and kernel-based learning methods, [36, 37] and [20] developed a Bayesian framework for SVMs and least square support vector machines (LSSVMs), respectively, with the guidance of the evidence framework. On the other hand, [50] showed that for a Bayesian neural network, if independent Gaussian distributions are used as the priors for network weights and bias, with the number of hidden units increasing, the prior over the network output functions converges to a Gaussian process (GP). A GP is a collection of random variables with the property that the joint distribution of any finite subset of these variables is Gaussian. If we consider the characteristic similarity between the mapping from input nodes/data to hidden units in a neural network and the mapping from input data to a feature space conceptually embedded in a LSSVM, it is not surprising that under the Gaussian noise assumption, the mean of the posterior prediction made by a GP coincides with the optimum decision function made by an LSSVM ([10, 20]). GPs have also been shown to be very effective to capture both linear and nonlinear relationships. Furthermore, GPs have a very explicit probabilistic model. These facts motivated us to develop a new method for microarray gene expression data analysis based on a hierarchical Bayesian learning framework, kernel-imbedded Gaussian process (KIGP).

To build a method based on GPs and learn it under a Bayesian framework is an active research field ([59]). Many researchers have been trying to analyze microarray via
a linear or generalized linear discriminative model with a hierarchical Bayesian structure, e.g. [38] and [81]. The effort to introduce a kernel-induced hierarchical Bayesian approach to the gene selection problem in a microarray analysis application is also emerging. For example, [9] developed a gene selection algorithm with a GP as the core discriminative learning function. They built the model by embedding an automatic relevance determination (ARD) parameter for each gene and adopted an “ordinal regression” model. Nonetheless, the corresponding kernel adopted in [9] is essentially a linear function. Besides, an ordinal regression does not necessarily provide optimum performance for a multi-classification problem. To this end, our goal is to build a more generalized kernel-induced hierarchical Bayesian model, through which we can relatively improve microarray analysis performance especially to a noisy (or non-trivial) dataset comparing to other benchmark methods, whereas we can also appropriately discriminate the genes that truly have significant importance for the target cancer types.

1.2.2 Feature Space and Kernel-Induced Learning Methods

Over the past decade, based on the results from statistical learning theory ([72]), the kernel-induced SVM was developed and intensively studied. It has been successfully applied to a number of learning tasks and is widely accepted as one of the state-of-the-art learning methods. [7] is a well regarded tutorial on the basic concepts in SVMs. An LSSVM differs from a SVM due to the use of different constraint and loss functions. It has also been successfully applied to many real classification applications ([35]). [48] provides a comprehensive review of kernel-induced learning methods. [10] and [63]
cover this research field, introducing the basic concepts and discussing advanced researches in kernel related learning methods.

We shall first introduce the general concept of feature space. Let us use \( \Psi(\bullet) \) to denote the function that maps from observation space (also called primal space) to feature space (also called dual space). The concept is that, in a proper feature space, the target of interest for a learning problem can be better represented, hence improving the analyzing performance. Fig. 1-2 shows a heuristic example. From this figure, one can see the two target classes in the observation space (left panel) are not linearly separable. But after the feature mapping, in the feature space (right panel), the two classes can be well separated by a straight line, therefore a linear method can be applied in the feature space to achieve a favorable result.

If the mapping function is the identity function, the feature space is the observation space itself. Therefore, most linear methods are actually trained in the observation space. For those state-of-the-art methods referenced in the last subsection, most of them can be categorized as learning in a feature space. For example, the feature space of a SPCA method ([5]) is the span of the first few principle components of the data; the random forest method ([14]) assumes the significant gene data follow a forest structure; ssEAM ([78]) is built on the condition that the significant data can be modeled well by a featured geometric representation, called “category”. As for a kernel-induced learning method, it realizes training in feature space through its implicit kernel structure.
One of the beauties of a kernel-induced method is that the data virtually are analyzed in feature space, whereas it has the advantage of doing all numerical calculations only in observation space. Given two vectors \( x \) and \( x' \) defined in observation space, then \( \Psi(x) \) and \( \Psi(x') \) represent the two mapped vectors in feature space. The value of the kernel function with respect to \( x \) and \( x' \) conceptually corresponds to the inner product of \( \Psi(x) \) and \( \Psi(x') \) in the feature space. That is,

\[
K(x, x') = \langle \Psi(x), \Psi(x') \rangle
\]  

(1-8)

Mercer’s theorem ([47]) provides a characterization of what kind of function can be a kernel function. That is, the function \( K(\bullet, \bullet) \) defined in the observation space is a kernel function if and only if the matrix \( K \) is positive semi-definite (has no negative eigenvalue), where the matrix \( K \) is defined by

\[
K = (K(x_i, x_j))_{i,j = 1}^\ell
\]  

(1-9)
\( x_1, x_2, \ldots, x_n \) are the set of observations. The matrix \( K \) is often called the Gram matrix or the kernel matrix.

On a side note, kernel-induced SVMs have been generally regarded as a tool for many classification and regression problems because their performance can approximately achieve the Bayesian bound without being given a precise probabilistic generative model. In theory, by exploring the close relationship between misclassification rate of SVMs and the generalized comparative Kullback-Leibler (GCKL) distance ([73]), [40] and [41] showed that if the regularization parameter is appropriately chosen and the dimension of the feature space is big enough, the solution of a SVM can approach the Bayesian bound when the training sample size is sufficiently large. This optimality property of SVM is another reason for the success of kernel-induced SVMs.

1.2.3 Markov Chain Monte Carlo Method

The major challenge for Bayesian analysis is to calculate Bayesian posterior distributions, which more often than not are analytically intractable in a real application. A lot of numerical algorithms have been introduced for this problem and the Markov chain Monte Carlo (MCMC) method is one of the most popular as well as successful approaches. The Gibbs sampler was first used in the “Boltzmann machine” neural network by [1], and it is perhaps also the simplest and the most widely used MCMC method. Since Gibbs sampler plays one of the major roles in this study, we shall describe it in more detail in this subsection.
In general, let $\mathbf{\theta} = [\theta_1, \theta_2, \ldots, \theta_J]^T$ to be the set of the model parameters and $\mathbf{D}$ be the training dataset, $\mathbf{D} = \{(x_i, y_i) \mid i = 1, 2, \ldots, n\}$. The general Bayesian analysis method essentially involves calculating the posterior expectation

$$E[a \mid \mathbf{D}] = \int a(\mathbf{\theta}) P(\mathbf{\theta} \mid \mathbf{D}) d\mathbf{\theta}$$

(1-10)

where $a(\mathbf{\theta})$ can be any function conditioned on the model parameters $\mathbf{\theta}$. This expectation, however, is difficult to integrate in most practical models. Alternatively, a numerical way to calculate the expectation in (1-14) is to use Monte Carlo integration by

$$E[a \mid \mathbf{D}] \approx \frac{1}{N} \sum_{i=1}^{N} a(\mathbf{\theta}^{[i]})$$

(1-11)

where $\mathbf{\theta}^{[1]}, \mathbf{\theta}^{[2]}, \ldots, \mathbf{\theta}^{[N]}$ are independently drawn from $P(\mathbf{\theta} \mid \mathbf{D})$. When $N$ is large enough, this approximation converges to its analytical integral under very general conditions.

Formula (1-11) is straightforward, but it is often infeasible to generate such an independent series $\mathbf{\theta}^{[1]}, \mathbf{\theta}^{[2]}, \ldots, \mathbf{\theta}^{[N]}$ when the posterior distribution $P(\mathbf{\theta} \mid \mathbf{D})$ is complicated. Nonetheless, it may be possible to generate a series of dependent values via a Markov chain that has $P(\mathbf{\theta} \mid \mathbf{D})$ as its stationary distribution and Monte Carlo integration still gives an unbiased estimate for $E[a \mid \mathbf{D}]$ after the Markov chain converges ([50, 60]). The methods built under this guideline are called Markov chain Monte Carlo (MCMC). Gibbs sampler is a special form of MCMC method.

Suppose there are $J$ parameters in the model, a Gibbs sampling algorithm is briefly described below ([19]):

1. Choose arbitrary starting values: $\mathbf{\theta}^{[0]} = [\theta_1^{[0]}, \theta_2^{[0]}, \ldots, \theta_J^{[0]}]$. 
2. Start at $l=1$ and complete the single cycle by drawing values from the $J$ distributions given by:

- $\theta_1^{[1]} \sim P(\theta_1 | D, \theta_2^{[1]}, \theta_3^{[1]}, ..., \theta_j^{[1]}, \theta_j^{[1-1]})$
- $\theta_2^{[1]} \sim P(\theta_2 | D, \theta_1^{[1]}, \theta_3^{[1]}, ..., \theta_j^{[1]}, \theta_j^{[1-1]})$
- $\theta_3^{[1]} \sim P(\theta_3 | D, \theta_1^{[1]}, \theta_2^{[1]}, ..., \theta_j^{[1]}, \theta_j^{[1-1]})$
- ... 
- $\theta_{j-1}^{[1]} \sim P(\theta_{j-1} | D, \theta_1^{[1]}, \theta_2^{[1]}, ..., \theta_{j-2}^{[1]}, \theta_{j-1}^{[1-1]})$
- $\theta_j^{[1]} \sim P(\theta_j | D, \theta_1^{[1]}, \theta_2^{[1]}, ..., \theta_{j-2}^{[1]}, \theta_{j-1}^{[1]})$

3. Set $l = l + 1$ and repeat the second step until convergence. (1-12)

Once convergence is reached, the conditional distributions from the simulation (1-12) contain sufficient information to reach the true posterior distribution of interest. Thus, we can approximately calculate $E[a | D]$ in (1-11) by $\frac{1}{L} \sum_{l=1}^{L} a(\theta^{[l]})$ with sufficiently large $L$, where $\theta^{[l]}$ is the $l$-th sample drawn from the Gibbs sampler (1-12) within each iteration after convergence.

### 1.2.4 Gaussian Processes for Microarray Analysis

The model of Gaussian Processes is one of the classical problems in probability and random process theory. It was intensively studied in the computer science field when researchers, such as in [49, 50], applied a Bayesian framework to learn a neural network, which is famously known as a machine learning method with superior performance.
Good introductions to Gaussian Processes can be found in such as [21], [49], [59] and the references therein.

In general, a continuous stochastic process is known as a collection of random variables, and each of these random variables takes real values from a probability distribution function. If we consider the output of a learning function \( g(e) \), where \( g \) is chosen according to a distribution \( D \) defined over a class of real-valued functions, the collection of such outputs is a stochastic process. The distribution \( D \) presents the prior belief in the likelihood that different functions will provide to the learning problem. A Gaussian process is a continuous stochastic process such that the marginal distribution for any finite subset of the collection of its output is a zero mean Gaussian distribution.

Specifically for the gene-selection problem defined in (1-5), \( t_i = g(x_{ji}) \), where \( x_{ji} = [x_{ji1}, x_{ji2}, \ldots, x_{jin}] \), \( i = 1,2, \ldots, n \), we assume

\[
P_{p \sim D}([g(x_{j1}), g(x_{j2}), \ldots, g(x_{jn})] = [t_1, t_2, \ldots, t_n]) \propto \exp(-\frac{1}{2}t'K^{-1}t),
\]

where \( K_{ij} = K(x_{ji}, x_{ji}) \), \( i, j = 1, 2, \ldots, n \). (1-13)

In (1-13), \( K(\cdot, \cdot) \) is the kernel function and \( K_\gamma \) is the kernel matrix introduced in subsection 1.2.2. The vector form for (1-5) is \( z = t + e + b1_n \). Under white Gaussian noise assumption for the noise \( e \), it consequently yields

\[
P(z \mid t) \propto \exp(-\frac{1}{2}(z - t - b1_n)'\Omega^{-1}(z - t - b1_n)), \text{ where } \Omega = \sigma^2I_n
\]

(1-14)

In (1-14), \( 1_n \) denotes an \( n \) by \( 1 \) vector, all elements of which are equal to \( 1 \); \( I_n \) represents an \( n \) by \( n \) identity matrix. Following the Bayes rule, we have
\[ P(\tilde{r}, t | \tilde{x}, X, z, \gamma) = \frac{P(z | t, \tilde{x}, X, \gamma)P(\tilde{r}, t | \tilde{x}, X, \gamma)}{P(z | x, \gamma)} \propto P(z | t)P(\tilde{r}, t | \tilde{x}, X) \]  

(1-15)

where \( \tilde{r} \) is the output associated with the new selected predictor vector \( \tilde{x}_r \) given the training data matrix \( X_r \) and the latent variable \( z \), which is presumed to be corrupted by noise. With a given kernel function and assuming the intercept \( b \), the variance of noise \( \sigma^2 \) and the gene-selection vector \( \gamma \) are all provided. Plugging (1-13) and (1-14) into (1-15), integrating with respect to \( t \), we obtain the marginal distribution of \( \tilde{r} \) given \( \tilde{x}_r \), \( X_r \) and \( z \). It is a Gaussian distribution ([10]):

\[ \tilde{r} | \tilde{x}, X, z, \gamma \sim N(f(\tilde{x}_r, X_r, z), V(\tilde{x}_r, X_r, z)), \]  

where

\[ f(\tilde{x}_r, X_r, z) = (z - b1_n)'(K_r + \sigma^2 I)^{-1}k_r, \]

\[ V(\tilde{x}_r, X_r, z) = K(\tilde{x}_r, \tilde{x}_r) - k_r'(K_r + \sigma^2 I)^{-1}k_r, \]

\[ K_r_{ij} = K(x_r_i, x_r_j), \ k_r = K(\tilde{x}_r, x_r), \ i, j = 1, 2, ..., n. \]  

(1-16)

The most popular kernel types include Gaussian kernel, polynomial kernel and sigmoidal kernel. If we use the input space as the feature space, we imply a linear kernel. The definitions of Gaussian kernel, polynomial kernel and linear kernel are provided in formula (2-1).

1.2.5 Illustrative Examples

To illustrate how different kernels (or a single kernel with different parameters) imply different feature space in a GP learning problem, we shall present three examples in this subsection. The GP method is described in the last subsection and we only
consider the mean of a GP for these examples. The first example is a one-dimensional regression problem, where the target function is non-linear. The second example is a classification problem with two-dimensional observations, where the Bayesian classifier takes a very non-linear form. The third example is similar to the second one except that the optimum class decision function can be well approximated by a linear function. From all these three examples, we shall show that a GP learner can have very different performance with applying different kernel functions. Also, with a proper kernel, a GP can perform almost as well as the relative optimum classifier (Bayesian classifier).

![Reconstructed functions of Example 1](image)

**Fig. 1-3: Reconstructed functions of Example 1.** $r$ denotes width of the applied Gaussian kernel.
Example 1: A non-linear regression problem

In this example, the true underlying function between the input (labeled by $x$) and the output (labeled by $y$) is $y = (x-5)^2/10 - \cos(x) + e^{-x}$. There are totally 20 observed samples, each of which is corrupted by an additive Gaussian noise (with 0 mean and unit variance). The observed samples and the true function are shown in Fig. 1-3 (the dots and the solid line respectively). The objective is to reconstruct a function of $x$ within the interested region ($0 \leq x \leq 10$), such that we can minimize the mean square error (MSE) between the regressed values (mean of the output of a GP) and the true values of the training sample set.

![MSE plot of Example 1](image)

Fig. 1-4: **MSE plot of Example 1.** $r$ denotes width of the applied Gaussian kernel. MSE represents “Mean Square Error”.
We applied the GP regression method with a Gaussian kernel to the training set. In each simulation, we chose a different width. The MSEs of the simulations with different width are drawn in Fig. 1-4. Some of the reconstructed functions are provided in Fig. 1-3. From Fig. 1-4, we can see that the GP regression with a Gaussian kernel delivered the best performance when $r^2$ is set around 5 (dotted line in Fig. 1-2). From Fig. 1-3, it can be easily observed that, with a small width, the reconstructed function is very localized around the observed training samples, leading an over-fitting (dot-broken line in Fig. 1-3). When the width increases, the reconstructed function becomes smoother.
and smoother. When the width is large enough, the reconstructed function almost
performs like a linear function (broken line in Fig. 1-3).

Example 2: A non-linear classification problem

In this example, the samples are two-dimensional and they consist of several
clusters. The generative model for the observation data is same as the one used in the
examples in section 2.3.3.2 for generating the significant gene data, except that cluster 1
and 2 are labeled as class “1” and the remaining clusters are labeled as class “-1”. Based
on this model, the optimum decision function with respect to the two dimensions is
shown as the solid lines in Fig. 1-5. With the same generative model, we generated an
independent testing set (with 5000 samples). Obviously, the linear method cannot work
for this example.

We then applied the GP learner with a Gaussian kernel to this training set. In each
simulation, we used a different kernel width. In each figures in Fig. 1-5, the red dots
(class 1) and the green dots (class -1) mark the output decision for the testing samples
based on the GP learner with different kernel widths. In Fig. 1-6, we show the
misclassification rate (MR) of the testing set in each simulation. As a comparison, we
also give the MR with the optimum decision function based on the generative model
(Bayesian bound) and the MR with the optimum linear decision function based on the
training set. We can see that, the GP learner with a Gaussian kernel gives its best
performance when \( \sigma^2 \) is set around 0.2. The MR in this simulation is fairly close to the
Bayesian bound.
From Fig. 1-5, we also can observe the same phenomena as illustrated in Example 1. When the kernel width is very small, the output class decision function by a GP is very localized around the span of the training set, which is very non-linear (say $r^2 = 0.01$). When the width increases, the decision function becomes smoother (say $r^2 = 0.1, r^2 = 1$). When the width is relatively large, the decision function performs very much like a linear function (say $r^2 = 10$).
Example 3: A linear classification problem

The training set and the testing set of this example are almost exactly the same as in Example 2, except that we applied a different label scheme in the generative model (Fig. 1-7). As the result, the optimum decision function can be well approximated by a linear function (solid lines in Fig. 1-7) in this example.

We then again applied the KIGP method with a Gaussian kernel to this training set. In each simulation, we used a different kernel width and the results are shown in Fig. 1-7 and Fig. 1-8. We can see that the GP learner with a Gaussian kernel gives its best performance for this example when \( r^2 \) is set around 1.

![Class decision function plot of Example 3](image)

**Fig. 1-7: Class decision function plot of Example 3.** In each of the figures, the asterisks and the circles denote the training samples in class “1” and class “-1”, respectively. The solid lines display the optimum decision function for this example. The red dots and green dots mark the testing samples in class “1” and “-1”, respectively, labeled by the GP learner. \( r \) denotes width of the applied Gaussian kernel.
In order to demonstrate how different kernel implies different feature space, hence delivering different generalized prediction performance, we draw the MR comparison plot with respect to different Gaussian kernel width for both Example 2 (non-linear) and Example 3 (linear) in Fig. 1-8. When the optimum decision function takes a very non-linear form, the fitted kernel width is relatively small (at 0.2 in Example 2). On the contrary, the optimum Gaussian kernel width is larger (1.0) when the underlying Bayesian classifier can be well approximated by a linear function (Example 3).

Fig. 1-8: MR comparison plot of Example 2 and 3. \( r \) denotes width of the applied Gaussian kernel. MR represents “Misclassification Rate”.
1.3 General Framework of This Study

In summary, this work is built upon a hierarchical Bayesian model. We focus on supervised learning problems, especially on cancer classification problems via microarray analyses. The model formulation for a binary-classification problem is given in subsection 1.2.1. The diagram of the general framework adopted in this study can be sketched as in Fig. 1-9, in which the overall data flow is described as follows:

1) Selecting the important genes from the input data via the gene-selection vector;

2) Conceptually mapping the selected data into the feature space and train the problem via an optimum linear classifier. With the kernel-induced learning theory, this can be done through a proper kernel function. The candidate training methods include such as SVM, LSSVM, GP, PLR and kernel Fisher discrimination (KFD). In this study, we only focus on the Gaussian process model introduced in subsection 1.2.4.

3) Hierarchically outputting the results with, in turn, the optimum kernel type and its associated optimum kernel parameter(s); the estimation of model parameters with the output optimum kernel function provided by the last level; and predictions for the testing samples (if applicable).
Fig. 1-9: Flow chart for the general framework of this study

The rest of this dissertation is structured as follows.

In chapter 2, by introducing a GP into the gene selection problem with a probit regression setting, we develop a Bayesian framework for analyzing microarray datasets. Specifically, the derivation for the Bayesian inference and the overall algorithm for a binary classification problem are given in section 2.1 (most contents of this section can be found in [79]); the extension to a multi-classification problem is found in section 2.2; the experimental results are provided in section 2.3.

In chapter 3, for a supervised learning problem, we propose a viable approach to building a kernel in accordance to the properties of the training data, named by natural kernel. The basic concepts of a general natural kernel method are outlined in section 3.1. Our proposed natural kernel based KIGP analysis algorithm is described in section 3.2. Simulation results are given in section 3.3.
In chapter 4, via embedding the KIGP framework into a general reversible jump Markov chain Monte Carlo (RJMCMC) algorithm, we present a unifying approach to simultaneously comparing performances of different kernel types. The general RJMCMC algorithm is shown in section 4.1. The RJMCMC-imbedded KIGP is discussed in section 4.2. Section 4.3 concludes this study with a unifying algorithm, such that we can realize an automated learning machine for microarray analysis. Verifying simulation examples are provided in the end of this section.

In the last chapter, we summarize this study, addressing the conclusions and some of the potential future research directions.
Chapter 2

KIGP Framework for Microarray Analysis

In this chapter, we outline a hierarchical Bayesian framework based on kernel-imbedded Gaussian processes (KIGPs) via a probit regression setting, in order to analyze a given tumor classification problem with the relative microarray gene profiling. Throughout this chapter, we assume that a kernel type is given and we shall discuss the kernel type competition problem in chapter 4. The hierarchical Bayesian model KIGP is developed via a probit regression setting based on the Gaussian Process model in subsection 1.2.4 and. A cascading adaptive algorithm with a Gibbs sampler as the core is designed to find appropriate kernel parameters (for a parametric kernel type only), to discover potentially significant genes and to make optimum predictions. It yields a classification problem with given microarray gene profiling. The algorithm for a binary classification problem is provided in section 2.1. The extension to a multi-classification problem is discussed in section 2.2. In section 2.3, we show a few simulated examples to illustrate the key elements and the effectiveness of the proposed KIGP framework. These examples demonstrate that, either in a binary-classification problem or in a multi-classification problem, the KIGPs can perform very close to the Bayesian bound without knowing the underlying generative probabilistic model in both the cases with linear Bayesian classifiers and those with very non-linear Bayesian classifiers. This sheds light on its promising potential to solve problems, for which linear methods are not effective.
We also report the results by applying KIGP to six published datasets (Table 2-2) and demonstrate that KIGP performs better or at least as well as any referred state-of-the-art microarray analysis method.

The most widely used kernel types in previous works include Gaussian kernel (GK) and polynomial kernel (PK). If input space is used as feature space, one implies using a linear kernel (LK). The kernel types used in this chapter are defined below (assuming \( x \) and \( x' \) are two vectors in observation/input space):

\[
\text{Linear kernel: } K(x, x') = \langle x, x' \rangle
\]

\[
\text{Polynomial kernel: } K(x, x') = (\langle x, x' \rangle + 1)^d
\]

where \( d = 1, 2, \ldots \) is the degree parameter.

\[
\text{Gaussian kernel: } K(x, x') = \exp\left(-\frac{\|x - x\|^2}{2r^2}\right)
\]

where \( r > 0 \) is the width parameter.

In (2-1), \( \langle *, * \rangle \) denotes the inner product between two vectors and \( \|\bullet\|^2 \) denotes 2-norm of the distance between two vectors. Throughout this study, we shall refer to a linear kernel as LK; to a polynomial kernel with degree \( d \) as PK(\( d \)); and to a Gaussian kernel with width \( r \) as GK(\( r \)).

In this study, we consider a classification problem as detailed in subsection 1.2.1. That is, there are \( n \) training samples. In each of the training samples, there are \( p \) investigated genes and a response class label denoted by \( y = [y_1, y_2, \ldots, y_n]' \). For a binary classification problem, we define

\[
y_i \in \{-1, 1\} \quad i = 1, 2, \ldots, n
\]
For a multi-classification problem with $M$ classes, we define

$$y_i \in \{1,2,...,M\} \quad i = 1,2,...,n$$

(2-3)

where "$M$" represents the label of the base class. The definitions of the training data matrix $x$, gene-selection vector $\gamma$, and its associate significant gene matrix $x_i$ are the same as in formula (1-1), (1-3) and (1-4), respectively.

### 2.1 KIGP Framework for a Binary Classification Problem

For a binary classification problem with a probit regression setting, the model is described in (1-6), where $e_i$ denotes the independent noise term and $e_i \sim N(0,\sigma^2)$; $b$ represents the intercept; and the link function $g(\bullet)$ is chosen from a class of real-valued functions, the output of which is assumed to be a Gaussian process. In (1-6), assuming that a kernel type is given and its associated model parameter set is termed as $\theta$, besides the latent variable vector $z$, we further introduce the random vector $t= [t_1,t_2,...,t_n]'$, where $t_i = g_\theta(X_{\gamma,i})$ for $i = 1,2,...,n$, and $e = [e_1,e_2,...,e_n]'$. This forms a hierarchical model that we represent by a directed acyclic graph in Fig. 2-1.

![Directed acyclic graph for Bayesian analysis of KIGP in a probit regression setting](image)

**Fig 2-1:** Directed acyclic graph for Bayesian analysis of KIGP in a probit regression setting. All the symbols are defined in Model (1-6); $\theta$ denotes the kernel parameters.

32
2.1.1 Prior Specifications

The priors for the parameters in this study are listed below:

(1) $\gamma_j$ is assumed to be a priori independent for all $j$, and

$$\Pr(\gamma_j = 1) = \pi_j, \text{ for } j = 1, 2, ..., p,$$

where the prior probability $\pi_j$ ranges between 0 to 1, which reflects prior knowledge of the importance of the $j$th gene.

(2) A non-informative prior is applied for the intercept $b$:

$$P(b) \propto 1.$$ (2-4b)

This is not a proper probability distribution function (PDF), but it leads to a proper posterior PDF.

(3) We apply the conjugate prior, inverted Gamma (IG) distribution, for the variance of noise $\sigma^2$. That is,

$$P(\sigma^2) \sim IG(\alpha, \beta)$$ (2-4c)

In this study, we set $\alpha = \beta = 0.1$. A practical issue on this prior set will be further discussed in subsection 2.5.1.

(4) Since the width parameter of a GK, $r$, basically is a scale parameter, inverted Gamma distribution is also a proper prior for $r$. Because too small or too big an $r$ will lead all elements of the corresponding Gaussian kernel matrix to be extremely close to 0 or 1 respectively, leading to unstable systems, a non-informative prior is not appropriate. Hence, throughout this study, we apply $IG(1,1)$ as the prior for $r^2$, that is

$$P(r^2) \sim IG(1,1)$$ (2-4d)
(5) For the degree parameter of the polynomial kernel, we assume a uniform distribution for the degree. To avoid possibly overfitting, throughout this section, we only consider the polynomial kernels with degree 1 or 2.

2.1.2 Derivation of the Computational Implementation of a Gibbs Sampler

With the model and the prior assumptions given above, we design a Gibbs sampler, through which we can sample the model parameters from their joint posterior distribution. As label \( y \) only depends on latent variable \( z \), all other model parameters are conditionally independent of \( y \) if \( z \) is given. For convenience, we drop the notation of the training set \( X \) in the following derivation and drop \( y \) as well when \( z \) is given. We assume the kernel type is given and its associated parameter is \( \theta \).

Lemma 2.1: For a model as given in (1-6) where prior for the output of the link function is a Gaussian process as formulated by (1-13), we have \( z \sim N(b_1, K_\gamma + \Omega) \) and \( t | z \sim N((I_n - \Omega(\Omega + K_\gamma)^{-1})(z-b_1), \Omega - \Omega(\Omega + K_\gamma)^{-1}\Omega) \), where \( \Omega = \sigma^2 I_n \) and \( K_{\gamma,i,j} = K(x_i, x_j) \) for \( i, j = 1, 2, ..., n \).

\[ t \sim N(0, K_{\gamma}), \quad z | t \sim N(t + b_1, \Omega) \]

where \( \Omega = \sigma^2 I_n \) and \( K_{\gamma,i,j} = K(x_i, x_j) \) for \( i, j = 1, 2, ..., n \).

Proof: For convenience, we drop the notation of all given parameters, \( b, \sigma^2 \) and \( \gamma \), in the following derivations. Under our KIGP model, we have

\[ t \sim N(0, K_{\gamma}), \quad z | t \sim N(t + b_1, \Omega) \]

where \( \Omega = \sigma^2 I_n \) and \( K_{\gamma,i,j} = K(x_i, x_j) \) for \( i, j = 1, 2, ..., n \).

The joint distribution of \( z \) and \( t \) can be formulated as:

\[
P(z, t) = P(z | t)P(t) \propto \exp\left\{ -\frac{1}{2} [(z - t - b_1)\Omega^{-1}(z - t - b_1) + t'K_n^{-1}t] \right\}
\]

\[
\propto \exp\left\{ -\frac{1}{2} [(t - \mu_\gamma)'(K_\gamma^{-1} + \Omega^{-1})(t - \mu_\gamma) + (z - b_1)'\Omega^{-1}(z - b_1) - \mu_\gamma'(K_\gamma^{-1} + \Omega^{-1})\mu_\gamma] \right\}
\]

34
where \( \mu_t = (\Omega^{-1} + K_\gamma^{-1})^{-1} \Omega^{-1} (z - b 1_n) \).  

(2-6)

In principle, if \( z \) and \( t \) form a joint Gaussian distribution, then both the marginal distribution of \( z \) and the conditional distribution of \( t \) given \( z \) are also Gaussian. Making use of the following equation from [6]:

\[
(A + C)^{-1} = A^{-1} - A^{-1} (A^{-1} + C^{-1})^{-1} A^{-1},
\]

(2-7)

it consequently yields

\[
P(z) = \int P(z, t)dt \propto \exp\left\{-\frac{1}{2} [(z - b 1_n)' \Omega^{-1} (z - b 1_n) - \mu_t' (K_\gamma^{-1} + \Omega^{-1}) \mu_t] \right\}
\]

\[
= \exp\left\{-\frac{1}{2} [(z - b 1_n)' (K_\gamma + \Omega)^{-1} (z - b 1_n)] \right\}
\]

(2-8)

and

\[
P(t \mid z) \propto \exp\left\{-\frac{1}{2} [(t - \mu_t)' (\Omega^{-1} + \Omega^{-1}) (t - \mu_t)] \right\}
\]

\[
= \exp\left\{-\frac{1}{2} [(t - \mu_t)' (\Omega + (K_\gamma + \Omega)^{-1})^{-1} (t - \mu_t)] \right\}
\]

(2-9)

where \( \mu_t = (I_n - \Omega (\Omega + K_\gamma)^{-1}) (z - b 1_n) \).

Or strictly,

\[
z \sim N(b 1_n, K_\gamma + \Omega)
\]

\[
t \mid z \sim N((I_n - \Omega (\Omega + K_\gamma)^{-1}) (z - b 1_n), \Omega - \Omega (\Omega + K_\gamma)^{-1} \Omega)
\]

(2-10)

It is straightforward to show that matrix \( \Omega - \Omega (\Omega + K_\gamma)^{-1} \Omega \) is non-negative definite.

(I) Sampling from \( \gamma \mid z, b, \sigma^2, \theta \)

We drop the notation of \( b, \sigma^2 \) and \( \theta \), taking them as given. With (2-10) and the prior given by (2-4-a), we have
\[
P(\gamma | z) \propto P(z | \gamma) \times P(\gamma) \propto \left[ \det(K_\gamma(\theta) + \Omega) \right]^{-\frac{1}{2}} \times \\
\exp\left\{ -\frac{1}{2} \left[ (z - b1_n)'(K_\gamma(\theta) + \Omega)^{-1}(z - b1_n) \right] \right\} \prod_{j=1}^{p} \pi_j^{\gamma_j} (1 - \pi_j)^{1 - \gamma_j}
\]

where \( \Omega = \sigma^2 I_n \); and \( K_\gamma(\theta) \) is defined as \( K_\gamma = k_\theta(x_a, x_b) \), which is the same kernel function as that defined in (2-5) with parameter(s) \( \theta \) \hspace{1cm} (2-11)

In practice, rather than sampling \( \gamma \) as a vector, we sample it component-wise from

\[
P(\gamma_j | z) \propto \left[ \det(K_\gamma(\theta) + \Omega) \right]^{-\frac{1}{2}} \times \\
\exp\left\{ -\frac{1}{2} \left[ (z - b1_n)'(K_\gamma(\theta) + \Omega)^{-1}(z - b1_n) \right] \right\} \pi_j^{\gamma_j} (1 - \pi_j)^{1 - \gamma_j}
\]

for \( j = 1, 2, \ldots, p \), where \( \Omega = \sigma^2 I_n \); \( K_\gamma(\theta) \) is the same as defined in (2-11) \hspace{1cm} (2-12)

(II) Sampling from \( t | \gamma, b, z, \sigma^2, \theta \)

With (2-10), the conditional distribution \( P(t | z, b, \sigma^2) \) is Gaussian distributed:

\[
t | \gamma, b, z, \sigma^2, \theta \sim N((I_n - \Omega(\Omega + K_\gamma(\theta))^{-1})(z - b1_n), \Omega - \Omega(\Omega + K_\gamma(\theta))^{-1}\Omega),
\]

where \( K_\gamma(\theta) \) and \( \Omega \) are defined in Eq. (2-11). \hspace{1cm} (2-13)

(III) Sampling from \( z | t, b, \sigma^2, y \)

Given the label vector \( y \), from the model (1-6), the conditional distribution of \( z \) given \( t \) is a truncated Gaussian distribution, and we have the following for \( i = 1, 2, \ldots, n \):

\[
z_i | t_i, b, \sigma^2, y_i = 1 \propto N(t_i + b, \sigma^2) \quad \text{truncated at the left by } 0,
\]

\[
z_i | t_i, b, \sigma^2, y_i = -1 \propto N(t_i + b, \sigma^2) \quad \text{truncated at the right by } 0. \hspace{1cm} (2-14)
\]

(IV) Sampling from \( b | z, t, \sigma^2 \)

When \( z \) and \( t \) are both given, this is a simple ordinary linear regression setting with only the intercept term. Under the prior assumption given by (2-4b), it yields
\( \sigma^2 | z, t, b \sim IG(0.1 + n/2, 0.1 + ns^2 / 2) \), where \( s^2 = \frac{1}{n} \sum_{i=1}^{n} (z_i - t_i - b)^2 \) (2-16)

2.1.3 Kernel Parameters Tuning

One of the major advantages of kernel-based learning methods is that one can explore the non-linearity feature of the underlying model for a given classification or regression problem through different kernels. It is therefore crucial to discuss the issue of kernel parameter tuning. With the KIGP framework constructed above, this becomes fairly straightforward.

As in the last subsection, we denote the kernel parameter(s) as \( \theta \), which can be either a scalar (e.g. the width parameter for a GK or the degree parameter for a PK) or a vector. For algorithmic convenience, we work with the logarithm of the conditional likelihood for the parameter \( \theta \):

\[
L(\theta) = \log(P(z | \gamma, b, \sigma^2, \theta)) = -\frac{1}{2} \log(\det(\mathbf{K}_y(\theta) + \sigma^2 \mathbf{I}_n)) - \frac{1}{2} \mathbf{z}^\top (\mathbf{K}_y(\theta) + \sigma^2 \mathbf{I}_n)^{-1} (\mathbf{z} - \mathbf{b}) - \frac{n}{2} \log(2\pi)
\] (2-17a)

With a proper prior distribution for \( \theta \), \( P(\theta) \), we have:

\[
P(\theta | z, \gamma, b, \sigma^2) \propto \exp(L(\theta)) * P(\theta)
\] (2-17b)
where $L(\theta)$ is defined in (2-17a). In this section, we specifically focus on two groups of kernels: the polynomial kernel and the Gaussian kernel, as defined in (2-1b) and (2-1c) respectively. For GK, with the prior given in (2-4d), we apply the Metropolis-Hasting (MH) algorithm to draw the samples from the posterior distribution given by (2-17b). Details of MH can be found in [19]. For PK, we simply calculate the likelihood with respect to each $d$ by (2-17a), and then sample $d$ accordingly.

Theoretically, with a linear kernel, the proposed KIGP performs very close to most other classic linear methods, which have been very effective in many real applications. With a Gaussian kernel, generally speaking, within a moderate range, as the width parameter increases, the dimension of its representative feature space will accordingly decrease and a GK with a large width performs fairly close to a linear kernel. On the other hand, when the width decreases the dimension of the feature space increases, and the performance of the classifier in the observation space becomes very non-linear. We have demonstrated this property of a GK in the examples in chapter 1. For a polynomial kernel, when the degree increases, the non-linearity of a KIGP also increases, while when the degree is equal to 1, the only difference between it and a linear kernel is a constant. In short, for either class of kernel, different values of the kernel parameter represent different feature spaces and for some specific value, their performance is close to that of a linear kernel. Therefore, the posterior distribution of the kernel parameter gives a hint on what kind of feature space is more appropriate to the target problem with the given training samples.
2.1.4 Proposed Gibbs Sampler

With the derivation above, assuming a kernel type has been given, the Gibbs sampling algorithm is as follows:

1. Start with proper initial value \([\gamma^{[0]}, b^{[0]}, t^{[0]}, z^{[0]}, \sigma^{[0]}, \theta^{[0]}]\); then set \(i = 1\).

2. Sample \(z^{[i]}\) from \(z | t^{[i-1]}, b^{[i-1]}, \sigma^{z[i-1]}\) via (2-14).

3. Sample \(t^{[i]}\) from \(t | \gamma^{[i-1]}, b^{[i-1]}, z^{[i]}, \sigma^{2[i-1]}, \theta^{[i-1]}\) via (2-13).

4. Sample \(b^{[i]}\) from \(b | z^{[i]}, t^{[i]}, \sigma^{2[i-1]}\) via (2-15).

5. Sample \(\sigma^{2[i]}\) from \(b | z^{[i]}, t^{[i]}, b^{[i]}\) via (2-16).

6. Sample \(\gamma^{[i]}\) from \(\gamma | z^{[i]}, b^{[i]}, \sigma^{2[i]}, \theta^{[i-1]}\) via (2-12) component-wise.

7. Sample \(\theta^{[i]}\) from \(\theta | z^{[i]}, b^{[i]}, \sigma^{2[i]}, \gamma^{[i]}\) via (2-17-b).

8. Set \(i = i + 1\) and go back to the step 2 until the required number of iterations.

9. Stop

(2-18)

In step 2 of (2-18), we follow Robert's ([61]) optimal exponential accept-reject algorithm to draw from a truncated Gaussian distribution. After a suitable burn-in period, we can obtain the posterior samples of \([z^{[i]}, t^{[i]}, b^{[i]}, \sigma^{2[i]}, \gamma^{[i]}, \theta^{[i]}]\) at the \(i\)th iteration with the procedure described in (2-18). The core calculation of the proposed Gibbs sampler involves inverse the matrix \(K_\gamma + \sigma^2 I\). Because the kernel matrix \(K_\gamma\) is symmetric and non-negative definite, \(K_\gamma + \sigma^2 I\) is symmetric and positive-definite. Therefore the algorithm is theoretically robust and the Cholesky decomposition (CD, [59]), along with
its parallel implementation, can be used in practice to improve the efficiency of this algorithm. The total computation complexity of the Gibbs sampler within each iteration is $O(pn^3)$.

2.1.5 Overall Algorithm

With a kernel type given a priori, our algorithm basically adopts a cascading structure and is composed of three consecutive phases: “kernel parameter fitting phase” (for parametric kernel type only, of course), “gene selection phase” and “prediction phase”. Although from the classical Bayesian perspective we can involve all the parameters into the Gibbs sampler in all three phases, we suggest to fix the kernel parameters after “kernel parameter fitting phase”; and fix gene-selection vector after “gene selection phase”, since in most cases we are more interested in the area around the peak of the posterior probability density function (PDF) or probability mass function (PMF) of a parameter, especially for the kernel parameters and the gene-selection vector. This strategy will lead to a much faster convergence of the Gibbs sampler. For all three phases, we need to discard some proper number of iterations in a burn-in periods. Some dynamic monitoring strategies to track the convergence of an MCMC simulation can be found in [19] and the references therein.

A practical issue for applying the algorithm in (2-18) needs to be addressed here. We suggest fix variance parameter $\sigma^2$ during a “kernel parameter fitting phase” and a “gene selection phase” because a varied variance during these two phases can easily lead to too small or too large $\sigma^2$ due to overfitting, which may cause instability of the system. Hence, fixing variance will help the algorithm be much more numerically stable and
converge faster. Following the regular probit regression model, we set $\sigma^2$ equal to 1 (step 5 in (2-18)) in the first two phases and only involve it into the Gibbs sampler in a “prediction phase”. The details of each phase are described hereafter.

2.1.5.1 Kernel Parameter Fitting Phase

In this phase, our primary interest is to find the appropriate value(s) for the kernel parameter(s) of the given kernel type. Initially, we focus on two kernel types, the polynomial kernel and the Gaussian kernel. We first involve all model parameters (except $\sigma^2$), including gene selection vector and kernel parameter, into the simulation of the algorithm given by (2-18). After convergence, the obtained samples within each iteration of (2-18) are equivalently drawn from the joint posterior distribution of all the parameters. For a PK, since the degree parameter is a discrete number, we can simply take the degree with the highest appearance frequency of the samples. For a GK, we calculate the histogram of the sample values of the width parameter with some proper number of bins. Then we pass the obtained histogram series through a Gaussian smoother (similar to Gaussian kernel density estimation). Finally, we take the center of the bin with highest histogram counts as the best fit of the width.

2.1.5.2 Gene Selection Phase

After “kernel parameter fitting phase”, we fix the kernel parameter at its fitted value (mode of the posterior) and then continue to run the Gibbs sampler. We briefly
suggest an empirical Bayes approach to determine whether a gene is potentially significant based on the posterior samples.

In this study, we essentially follow the key concept introduced in [17] to assess whether or not a gene is of significant importance. First of all, we define a statistic named “Normalized Log-Frequency” (NLF) to measure the relative potential significance for each gene. If we denote by $F_j$ the frequency of the $j$th gene in the posterior samples, the definition of NLF is:

$$NLF_j = \frac{LF_j - \mu_L}{s_L}, \quad LF_j = \log(F_j) \quad \text{for} \quad j = 1, 2, ..., p$$

where $\mu_L = \frac{1}{p} \sum_{j=1}^{p} LF_j$, $s_L^2 = \frac{1}{p-1} \sum_{j=1}^{p} (LF_j - \mu_L)^2$ (2-19)

In (2-19), if $F_j$ is 0, simply set it as $1/2$ over the total number of iterations. The rationale for defining NLF as the key statistic is because the logarithm of a Gamma distribution can be well approximated by a normal distribution, while empirically the Gamma is a proper distribution for the frequency of the posterior samples of the genes from a homogenous group.

With NLFs, we consider a simple Bayes model. Suppose that the $p$ NLF-values fall into two classes, “insignificant” or “significant”, corresponding to whether or not $NLF_j$, for $j = 1, 2, ..., p$, is generated according to the null hypothesis, with prior probabilities $P_{b_0}$ and $P_{b_1} = 1 - P_{b_0}$, for the two classes, respectively; and that $NLF_j$ has the conditional prior density either $f_0(NLF)$ or $f_1(NLF)$ depending on its class. More formally:
\[ P_{b_0} = \Pr\{\text{Insignificant}\}, \ \Pr(NLF \mid \text{Insignificant}) = f_0(NLF) \]

\[ P_{b_1} = \Pr\{\text{Significant}\}, \ \Pr(NLF \mid \text{Significant}) = f_1(NLF) \]

(2-20)

We then define a term, the local "false discovery rate (fdr)"

\[ \text{fdr}(NLF) = f_0(NLF) / f(NLF) \]  

(2-21)

In (2-21), \( f(NLF) \) denotes the marginal distribution of NLF and \( \text{fdr}(NLF) \) is a precise estimator for the posterior probability of the null hypothesis (insignificant class) given the statistic NLF in most microarray analysis applications \( (P_{b_0} > 0.99) \). With \( \text{fdr}(NLF) \), we accordingly decide whether or not a target gene is "significant" with a confidence level.

To calculate \( \text{fdr}(NLF) \), one needs to properly estimate the marginal distribution of NLF, \( f(NLF) \), and to choose its density under null hypothesis, \( f_0(NLF) \). To estimate \( f(NLF) \), we apply the kernel density estimation method with a Gaussian kernel ([27]) to the ensemble values of the observed NLFs, \( \{ NLF_j, \ j = 1, 2, \ldots, p \} \).

As for the density of NLF for null hypothesis, \( f_0(NLF) \), the basic assumption we impose here is that the statistic NLF under null hypothesis is of a normal distribution. Since in most real microarray analysis problems, \( P_{b_1} \) is much smaller than \( P_{b_0} \) (say \( P_{b_0} > 0.99 \)), based on the definition in (2-19), it is safe to choose the standard normal (zero mean, unit variance) as \( f_0(NLF) \). Throughout this study, we always choose the standard normal as the density of NLF under null hypothesis.
2.1.5.3 Prediction Phase

After the "gene selection phase", the gene-selection vector is fixed. We continue to run the proposed Gibbs sampler (2-18) and the computational complexity of the proposed Gibbs sampler within each iteration dramatically decreases to $O(n^3)$. After a proper new burn-in period, we can draw samples of $z$, $b$, and $\sigma^2$ within each iteration in the "prediction phase". Following (1-16), the posterior PDF for the output $\tilde{t}$ given the test gene expression data $\tilde{x}$ in the $l$-th iteration is Gaussian distributed:

$$
\tilde{t}^{[l]} | \tilde{x}, X, z^{[l]}, b^{[l]}, \sigma^{2[l]} \\
\sim N(f^{[l]}(\tilde{x}, X, z^{[l]}, b^{[l]}, \sigma^{2[l]}), V(\tilde{x}, X, z^{[l]}, b^{[l]}, \sigma^{2[l]})) = N(f^{[l]}, V^{[l]}),
$$

where

$$
f^{[l]} = (z^{[l]} - b^{[l]}1_n)'(K_x + \sigma^{2[l]}1_n)^{-1}k_y, \\
V^{[l]} = K(\tilde{x}, \tilde{x}) - k_y'(K_x + \sigma^{2[l]}1_n)^{-1}k_y,
$$

$K_{y,j} = K(x_{y,i}, x_{y,j}), \; k_{y,j} = K(\tilde{x}, x_{y,j})$, for $i, j = 1, \ldots, n; \; l = 1, \ldots, L. \tag{2-22a}$

The predictive probability for the output label $\tilde{y}$ given $\tilde{x}$ then can be estimated by using Monte Carlo integration:

$$
P(\tilde{y} = 1 | X, y, \tilde{x}) = \frac{1}{L} \sum_{l=1}^{L} \Phi \left( \frac{f^{[l]} + b^{[l]}}{\sqrt{V^{[l]}}} \right), \\
P(\tilde{y} = -1 | X, y, \tilde{x}) = 1 - P(\tilde{y} = 1 | X, y, \tilde{x}),
$$

where $\Phi(x) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt \tag{2-22b}$

2.1.5.4 Discussion on the Simplified Procedures

A major drawback of using a KIGP is its computational complexity. This can be alleviated by applying a prescreening procedure for gene selection. There is little risk to
eliminate those genes that exhibit almost constant gene expression level across different class of samples. For example, we can do a preliminary gene selection based on the UR procedure suggested in [16] or via a Wilcoxon statistics as adopted in [13]. This procedure can dramatically decrease the computation cost and it would not lose predictive performance in most real applications for a KIGP.

Another approach to enhancing the algorithm efficiency is through proper approximation for the Bayesian inference. Within each iteration of the Gibbs sampler, rather than sticking to the strict analytical form for Bayesian inference, we also can apply a Laplace approximation to the posterior distraction for the latent variables. One algorithm based on this line to solve a GP-based model is provided in [76]. The bias of this procedure depends on whether or not the posterior distribution of the latent variables are unimodal and symmetric. We think, for most real problems, the assumption above should be a good approximation for the true distribution. Therefore this approach also can possibly decrease the computation complexity while not sacrificing much predictive performance.

### 2.2 KIGP Framework for a Multi-Classification Problem

The extension from a binary classification problem to a multi-classification problem in a probit regression setting is straightforward. For example, in [83], the authors extended a linear probit regression model for a binary classification problem to a multinominal linear probit regression model for a multi-classification problem to solve the gene-selection problem. Roughly following their line, in this subsection, we will extend
the KIGP model described in section 2.1 to a kernel-induced model that can handle with a
multi-class classification problem for analyzing microarray data.

Assuming we denote the label as in equation (2-3), we then introduce \( M - 1 \) latent
variables (termed by \( z \)), each of which specifies one of the target classes from the base
class. More formally, by introducing an \( n \times (M - 1) \) latent matrix \( z \) and \( t \), the built
model is formulated by:

\[
[z_m]_i = g_m(x_{ni}) + b_m + [e_m]_i = [t_m]_i + b_m + [e_m]_i \quad \text{for } i = 1, 2, \ldots, n, \ m = 1, 2, \ldots, M - 1
\]

such that \( y_i = \begin{cases} M & \text{if } [z_k]_i \leq 0 \\ k & \text{if } [z_k]_i > 0 \end{cases} \), where \( k = \arg \max_{m=1, \ldots, M-1} [z_m]_i \) \hspace{1cm} (2-23)

In formula (2-23), \( x_{ni} \) denotes the \( i \)-th row of the matrix \( X \); \( [e_m]_i \) symbolizes the noise
term that is assumed to be independent and identically distributed (IID) Gaussian with
zero mean and \( \sigma^2 \) variance; \( b_m \) represents the intercept; and \( g_m(.) \) is chosen from a
class of real-valued functions, the output of which is specifically assumed to be a
homogeneous Gaussian process. For convenience, we define the vector form as:

\[
z = [z_1, z_2, \ldots, z_{M-1}]', \ t = [t_1, t_2, \ldots, t_{M-1}]', \ e = [e_1, e_2, \ldots, e_{M-1}]', \ b = [b_1, b_2, \ldots, b_{M-1}]'
\]

Model (2-23) thus can be expressed in the vector form,

\[
z = t + e + b, \ m = 1, 2, \ldots, M - 1
\]

Apparently, when \( M = 2 \), model (2-23) is exactly same as the binary classification model
described in (1-6). Therefore, model (2-23) is a generalized version of model (1-6).

2.2.1 Derivation of the Computational Implementation of a Gibbs Sampler

The prior assumption for each classifier is the same as that adopted in the binary
classification model, thus the derivation for the Gibbs sampler for a multi-classification
problem is very similar to that in section 2.1. Again, for convenience, we drop the notation for the training set \( X \) and \( y \) (when \( z \) is given) in the following derivation.

(I) **Sampling from** \( \gamma \mid z, b, \sigma^2, \theta \)

We drop the notation of \( b, \sigma^2 \) and \( \theta \), taking them as given. Based on Lemma 2.1 and model (2-23), similar to the binary classification case, we have the “component-wise sampling form”:

\[
P(\gamma_j \mid z) \propto \left\{ \prod_{m=1}^{M-1} \left[ \det(K_{ym} + \Omega) \right]^{\frac{1}{2}} \right\} \exp\left\{ -\frac{1}{2} \sum_{m=1}^{M-1} \left[ (z_m - b_m 1_n)^T(K_{ym} + \Omega)^{-1}(z_m - b_m 1_n) \right] \right\} \pi_j^{\gamma_j(1-\pi_j)^{1-\gamma_j}} \quad j = 1, 2, ..., p 
\]

(2-24)

In (2-24), \( [K_{ym}]_{ij} = K_{ym}(x_i, x_j), \ i, j = 1, 2, ..., n ; \ \Omega = \sigma^2 I_n \).

(II) **Sampling from** \( t \mid \gamma, b, \sigma^2, \theta, z \)

The conditional distribution \( P(t_m \mid z_m, b_m, \sigma^2, \gamma) \) is Gaussian distributed:

\[
t_m \mid z_m, b_m, \sigma^2, \theta, \gamma \sim N((I_n - \Omega(\Omega + K_{ym})^{-1})(z_m - b_m 1_n), \Omega - \Omega(\Omega + K_{ym})^{-1} \Omega))
\]

for \( m = 1, 2, ..., M - 1 \), where \( K_{ym} \) and \( \Omega \) are defined in Eq. (2-24). (2-25)

(III) **Sampling from** \( z \mid t, b, \sigma^2 \)

Given the label vector \( y \), under the model in (2-23), the conditional distribution of \( z \) given \( t, b \) and \( \sigma^2 \) is truncated Gaussian distributed and can be drawn as follows:

For \( i = 1, 2, ..., n \)

If \( y_i = m, \ m \neq M \), first draw \( z_{im} \) from

\[
z_{im} \mid t_m, b_m, \sigma^2, y_i = m \propto N(t_{im} + b_m, \sigma^2) \text{ truncated left by } \max_{j \neq m}[0, z_{ij}, j = 1, 2, ..., M - 1]
\]

(2-26a)

Then draw \( z_{ij} \) for \( j = 1, 2, ..., M - 1 \), \( j \neq m \) from
\[ z_{ij} \mid t_{ij}, b_j, \sigma^2, y_i = m \sim N(t_{ij} + b_j, \sigma^2) \text{ truncated right by the new } z_m \text{ (from (2-26a))} \]  

(2-26b)

Otherwise, \( y_i = M \), simply draw \( z_{ij} \) for \( j = 1, 2, \ldots, M - 1 \) from

\[ z_{ij} \mid t_{ij}, b_j, \sigma^2, y_i = M \sim N(t_{ij} + b_j, \sigma^2) \text{ truncated right by 0} \]  

(2-26c)

(IV) Sampling from \( \mathbf{b} \mid \mathbf{z}, \mathbf{t}, \sigma^2 \)

When \( \mathbf{z} \) and \( \mathbf{t} \) are both given, this is an ordinary linear regression setting with an intercept term. Under the non-informative prior assumption given by (2-4b), it yields:

\[ b_m \mid \mathbf{z}, \mathbf{t}, \sigma^2 \sim N(\mu_m, \sigma^2 / n), \quad m = 1, \ldots, M - 1, \quad \mu_m = \frac{1}{n} \sum_{i=1}^{n} ([z_{im} - \lfloor t_{im} \rfloor]) \]  

(2-27)

(V) Sampling from \( \sigma^2 \mid \mathbf{z}, \mathbf{t}, \mathbf{b} \)

With \( IG(\alpha, \beta), \alpha > 0, \beta > 0 \), as the prior, the conditional posterior distribution for \( \sigma^2 \) is also an inverted gamma distribution. That is

\[ \sigma^2 \mid \mathbf{z}, \mathbf{t}, \mathbf{b} \sim IG(\alpha + n(M - 1)/2, \beta + s^2/2) \]

where \( s^2 = \sum_{m=1}^{M-1} \sum_{i=1}^{n} (\lfloor z_{im} \rfloor - \lfloor t_{im} \rfloor - b_m)^2 \)  

(2-28)

(VI) Sampling from \( \theta \mid \mathbf{z}, \mathbf{b}, \sigma^2, \gamma \)

We denote kernel parameter(s) as \( \theta = [\theta_1, \theta_2, \ldots, \theta_{M-1}] \), in which \( \theta_m \) represents kernel parameter(s) for classifier \( m \). \( \theta_m \) can be either a scalar or a vector. With a proper prior for the kernel parameter set \( \theta_m \), \( P(\theta_m), \quad m = 1, \ldots, M - 1 \) the posterior PDF for \( \theta \) is

\[ P(\theta_m \mid \mathbf{z}_m, \gamma, b_m, \sigma^2) \propto \exp(L(\theta_m)) \times P(\theta_m), \quad m = 1, \ldots, M - 1, \text{ where} \]
\[
L(\theta_m) = \log(P(z_m \mid \gamma, b_m, \sigma^2, \theta_m)) = -\frac{1}{2} \log(\text{det}(\mathbf{K}_m(\theta_m) + \sigma^2 \mathbf{I}_n)) - \frac{1}{2} \left[ (z_m - b_m \mathbf{1}_n)' (\mathbf{K}_m(\theta_m) + \sigma^2 \mathbf{I}_n)^{-1} (z_m - b_m \mathbf{1}_n) \right] - \frac{n}{2} \log(2\pi)
\]

(2-29)

In (2-29), \( \mathbf{K}_m(\theta_m) \) is defined in (2-24) as a function of the relative kernel parameter \( \theta_m \).

### 2.2.2 Proposed Gibbs sampler

With the derivation in the last subsection, we summarize the proposed Gibbs sampling algorithm as follows:

1) Start with a proper initial value \( [\gamma^{[0]}, \sigma^{2[0]}, b^{[0]}, t^{[0]}, z^{[0]}, \theta^{[0]}] \); then set \( i = 1 \)

2) Sample \( z^{[i]} \) from \( z \mid t^{[i-1]}, b^{[i-1]}, \sigma^{2[i-1]}, \theta^{[i-1]} \) via (2-26a), (2-26b) and (2-26c)

3) Sample \( t^{[i]} \) from \( t \mid \gamma^{[i-1]}, b^{[i-1]}, \sigma^{2[i-1]}, \theta^{[i-1]}, z^{[i]} \) via (2-25)

4) Sample \( b^{[i]} \) from \( b \mid z^{[i]}, t^{[i]}, \sigma^{2[i]} \) via (2-27)

5) Sample \( \sigma^{2[i]} \) from \( \sigma^2 \mid z^{[i]}, t^{[i]}, b^{[i]} \) via (2-28)

6) Sample \( \gamma^{[i]} \) from \( \gamma \mid z^{[i]}, b^{[i]}, \sigma^{2[i]} \) component-wise via (2-24)

7) If applicable, sample \( \theta^{[i]} \) from \( \theta \mid z^{[i]}, b^{[i]}, \sigma^{2[i]}, \gamma^{[i]} \) via (2-29)

8) Set \( i = i + 1 \) and go back to step 2 until meeting the required number of iterations

9) Stop

(2-30)

The discussions on the implementations of the Gibbs sampler, the "kernel parameter fitting phase" and the "gene selection phase" are identical to those for the binary classification problem. The details for the prediction phase of a multiclassification problem are slightly different and they are discussed in the next subsection.
2.2.3 Prediction

After the kernel parameter fitting phase and the gene selection phase, both kernel parameter(s) and gene-selection vector have been fixed. We continue to run the Gibbs sampler given by (2-30). After a proper new burn-in period, we can sample \( z, b, \sigma^2, \gamma \) and \( \theta \) (if applicable) within each iteration of the simulation. The yielded posterior predictive PDF for the interim hidden variable \( \tilde{t} \) of the new testing gene data \( \tilde{x} \) within each iteration is Gaussian distributed:

\[
\tilde{t}_m^{[l]} \mid \tilde{x}_m^{[l]}, X_m, z_m, \sigma_m^{[l]}, b_m^{[l]}
\sim N(f(\tilde{x}_m^{[l]}, X_m, z_m, \sigma_m^{[l]}, b_m^{[l]}), V(\tilde{x}_m^{[l]}, X_m, z_m, \sigma_m^{[l]}, b_m^{[l]})) = N(f_m^{[l]}, V_m^{[l]})
\]

\( m = 1, 2, \ldots, M - 1 \), \( l = 1, 2, \ldots, L \), where

\[
f_m^{[l]} = f(\tilde{x}_m^{[l]}, X_m, z_m, \sigma_m^{[l]}, b_m^{[l]}) = (z_m^{[l]} - b_m^{[l]}1_n)'(K_{x_m^{[l]}x_m^{[l]}} + \sigma_m^{[l]}I_n)^{-1}K_{x_m^{[l]}y_m^{[l]}},
\]

\[
V_m^{[l]} = V(\tilde{x}_m^{[l]}, X_m, z_m, \sigma_m^{[l]}, b_m^{[l]}) = K_m(\tilde{x}_m^{[l]}, \tilde{x}_m^{[l]}) - k_{y_m^{[l]}}(K_{y_m^{[l]}y_m^{[l]}} + \sigma_m^{[l]}I_n)^{-1}k_{y_m^{[l]}}
\]

\[
[K_{y_m^{[l]}y_m^{[l]}}]_{ij} = K_m(\tilde{x}_{y_m^{[l]}, i}, \tilde{x}_{y_m^{[l]}, j}), \quad k_{y_m^{[l]}} = K(\tilde{x}_{y_m^{[l]}, i}, \tilde{x}_{y_m^{[l]}, i}), \quad i, j = 1, 2, \ldots, n,
\]

(2-31a)

The predictive probability for the output label \( \tilde{y} \) given \( \tilde{x} \) can be estimated by using a Monte Carlo integration:

\[
P(\tilde{y} = m \mid X, y, \tilde{x}) = \frac{1}{L} \sum_{l=1}^{L} \delta(m - k_m^{[l]}), \quad m = 1, 2, \ldots, M,
\]

where \( \delta(m - k_m^{[l]}) = \begin{cases} 1 & \text{if } m = k_m^{[l]} \\ 0 & \text{otherwise} \end{cases} \), and

\[
k_m^{[l]} = \arg \max_{m \in \{1, \ldots, M - 1\}} \{ f_m^{[l]} + b_m^{[l]} \}, \quad \text{if } f_k^{[l]} + b_k^{[l]} \geq 0; \text{ otherwise } k_m^{[l]} = M.
\]

\( f_m^{[l]} \) is drawn from (2-31a) and \( b_m^{[l]} \) is drawn from algorithm (2-30).

(2-31b)
2.3 Simulation Results and Discussions

In some examples of this section, we compare the performance of a KIGP to that of several established kernel-related microarray data analysis methods such as GP_ARD ([9]), SVM and PLR. We also include PAM as a benchmark method. To implement GP_ARD, we relied on the C-package provided by the original authors ([9]) to select the significant genes. To implement SVM and PLR, we adopted both RFE ([26]) and UR ([16]) as the gene selection scheme, respectively. For simulations with other classifiers, we relied on KIGP algorithm within the prediction phase (with an LK) for classifier GP_ARD; function “svm” in R-package “e1071” for classifier SVM (using “one-against-all” strategy for a multi-classification problem); R-package “klr” provided by Dr. Zhu ([85]) for classifier PLR; and R-package “pamr” for classifier PAM.

2.3.1 Predictive Fit Measure Approaches to Model Performance Assessment

If sufficient independent testing samples are available for a learner, we can calculate its predictive fit measure to accurately assess the generalized performance of it. Popular predictive fit measures include misclassification rate (MR) or average predictive probability (APP) of the true labels. Throughout this study, we always refer APP to APP of the true labels.

Assuming that there are $W$ independent testing samples $\{ (\tilde{x}_i, \tilde{y}_i), \ldots, (\tilde{x}_W, \tilde{y}_W) \}$, where $\tilde{x}_i$ denotes the $i$-th testing microarray data and $\tilde{y}_i$ is its associated class label, $i = 1, 2, \ldots, W$, the MR for this testing set can be estimated by

$$MR_{\text{bin}} = \frac{1}{W} \sum_{i=1}^{W} MC_i,$$

where $MC_i = \begin{cases} 1 & \text{if } \tilde{y}_i \neq \tilde{y}_i \\ 0 & \text{if } \tilde{y}_i = \tilde{y}_i \end{cases}$.
\[ \tilde{y}_i = \arg\max_{m \in \{-1, 1\}} \{ P(\tilde{y}_i = m | X, y, \bar{x}_i, K) \} \text{ for a binary classification problem} \quad (2-32a) \]

\[ \tilde{y}_i = \arg\max_{m \in \{1, \ldots, M\}} \{ P(\tilde{y}_i = m | X, y, \bar{x}_i, K) \} \text{ for a multi-classification problem} \quad (2-32b) \]

And the APP for the testing set is defined by:

\[ APP_{\text{test}} = \frac{1}{W} \sum_{i=1}^{W} P(\tilde{y}_i = \tilde{y}_i | X, y, \bar{x}_i, K). \quad (2-33) \]

In both (2-32) and (2-33), the probability \( P(\tilde{y}_i | X, y, \bar{x}_i, K) \) is evaluated by (2-31a) and (2-31b) (or by (2-22a) and (2-22b) for a binary classification problem). Obviously, a better model should have a smaller MR and a higher APP.

In many real applications, independent testing samples are not available. To still use a predictive fit measure to assess the generalized performance of the output model, one can resort to a cross-validation (CV) procedure. In this study, we shall adopt a leave-one-out cross-validation (LOOCV) approach. In detail, after “kernel fitting phase” and “gene selection phase”, KIGP will deliver a fitted kernel function and a set of identified significant genes. Then for each of the training samples, we one by one treat it as the testing set and take the rest samples as the training set to complete “prediction phase”. This procedure will yield a predictive measure (say MR or APP) for each sample, the average of which will be the predictive measure for the output model via an LOOCV procedure.

The measure via the LOOCV procedure mentioned above obviously contains a so-called “gene selection bias” because LOOCV is applied after the gene selection phase. That means, the index of output significant gene set contains information from all
samples. As the result, although LOOCV is a good way to test the effectiveness and correctness of the selected significant gene set with the fitted kernel function, it is not consistent to assess the generalized performance of the underlying kernel type (or model). Therefore, when we focus more on the generalized performance of the prescribed kernel type while an independent testing set is not available, we shall apply a more reliable CV procedure, the multiple independent 3-fold CV procedure introduced in [16]. The detail of the procedure is as follows. For each CV, one first randomly partitions the training set into three sets with a balanced ratio of the class labels. Then for each part, one treats one of three sets as the testing set and applies KIGP to the remaining two sets as the training set, including all three phases of a KIGP, getting the predictive fit measure for this testing set. After running this procedure for all three sets, one gets the predictive measure for all samples. If one does multiple independent (in terms of dataset split) 3-fold CVs, the average of the yielded predictive measure will give a consistent assessment of the generalized performance of the given kernel type (model).

2.3.2 Simulated Examples

2.3.2.1 Examples with a Binary Classification Model

The first two examples were designed to illustrate the key concepts, elements and procedures of the KIGP framework. In the first example, the Bayesian classifier of the underlying generative model is a straight line; in the second one, the Bayesian classifier is very non-linear. We set the number of the significant genes to two so that we can better graphically display the Bayesian classifier and the relative performance of the proposed KIGP algorithm. For both cases, the number of training samples is 20, 10 of which were
generated from the class “1” and the other 10 samples were generated from the class “-1”; for each sample, the number of investigated genes is 200; the index of the two underlying explanatory genes was preset as [23, 57].

We further designed the third example to demonstrate the effectiveness of the KIGP method when the number of investigated genes is large, especially for a problem with a very non-linear Bayesian classifier. A total of 1000 genes in a simulated microarray experiment and 10 of them were preset as the significant genes with indices [64, 237, 243, 449, 512, 573, 783, 818, 890, 961]. Similar to the first two examples, the number of training samples is 20, 10 of which were generated from the class “1” and the other 10 samples were generated from the class “-1”; the number of testing samples is 5000.

**Example 1: Case with a linear Bayesian classifier**

In this case, for class “1”, the two preset significant genes were generated from a multivariate Gaussian $N(\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix})$; for class “-1”, these two genes were generated from $N(\begin{bmatrix} -1 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix})$. For the insignificant genes, each of them was independently generated from the standard normal $N(0,1)$. The probabilities of the two classes are equal. For testing, we independently generated 5000 samples with the underlying generative model. Obviously, the Bayesian classifier for these two classes is a linear combination of the two prescribed significant genes.

We applied the KIGP method with a PK and with a GK to the training set. As a benchmark comparison, we also applied KIGP with an LK to the training set (obviously,
there is no "kernel parameter fitting phase" for KIGP with an LK). The prior probability for $\gamma_j = 1$ in the Gibbs sampling simulations was set as 0.01. For all the Gibbs sampling simulations in this example, we ran 5000 iterations in both the "kernel parameter fitting phase" and the "gene selection phase" and considered the first 1000 iterations as a burn-in period. In the "prediction phase", we ran 2000 iterations and discarded the first 500 iterations as the burn-in period. Throughout this section, we report all the genes with local false discover rate (fdr) smaller than 0.05.

For the simulation with a GK, the posterior PDF of the width "r" is plotted in Fig. 2-2-a, in which the mode is found at around 1.61. Therefore, after the "Kernel parameter fitting phase", the kernel was fixed at GK(1.61). With the samples obtained in the "gene selection phase", the NLF for each gene was calculated (Fig. 2-3-c). Following the procedure described in the "Gene selection phase" subsection, the local fdr with respect to each NLF value is estimated (Fig. 2-2-b). Under a 0.95 confidence level, the cutoff value for NLF is 3.83 and only the two prescribed genes (23, 57) were found to be significant. The contours of the posterior predictive probability of the class "1" are drawn in Fig. 2-3-d, where the X-axis and the Y-axis present the values of the gene 23 and the gene 57, respectively; the numbers associated with contour curves are probabilities; the asterisks denote the positive training samples; the circles exhibit the negative training samples; the dotted line shows the Bayesian classifier. The MR of the independent testing is 0.028, while that of the Bayesian classifier (Bayesian bound) is 0.013. For the simulation with a PK, after the "kernel parameter fitting phase", the estimated posterior PMF for the degree, "d=1", is equal to 0.797. We thus fixed the kernel at PK(1) after this phase. With the similar gene-selection procedure as described in the simulation with a
GK, the two prescribed genes again were found as the only two significant genes (Fig. 2-3-e). The relative contour plot of the posterior predictive probability of the class "1" is found in Fig. 2-3-f. The MR of the independent testing is 0.017 for this simulation. One can see that the performance of KIGP with a PK(1) is very similar to that of a KIGP with an LK (Fig. 2-3-a and 2-3-b). Both performed very close to the Bayesian classifier.

**Fig. 2-2: The Interim results of the KIGP with a GK applied to the simulated examples for the binary classification model.** (a) and (b) are for the linear example; (c) and (d) are for the non-linear example. (a) The estimated marginal posterior PDF of the width (solid line) versus its prior PDF (dotted line), where the mode is at around 1.61. (b) The local fdr with GK(1.61) (with standard normal as the density of NLF under null hypothesis); the horizontal dotted line represents the threshold of the fdr (0.05); the vertical dotted line shows the resulted cutoff value for NLF (3.83). (c) The estimated marginal posterior PDF of the width, where the mode is at around 0.81. (d) The local fdr with GK(0.81) (with standard normal as the density of NLF under null hypothesis); the horizontal dotted line represents the threshold of the fdr (0.05); the vertical dotted line shows the resulted cutoff value for NLF (3.68).
Overall, KIGP with a GK, a PK or an LK all worked very well in this linear Bayesian classifier case. All of them found the two preset significant genes and performed very close to the Bayesian bound. Since KIGP with PK(1) was slightly better, we should use PK(1) and the two genes found by PK(1) to make any further prediction.
Example 2: Case with a non-linear Bayesian classifier

In this case, for class "1", the two preset significant genes (the gene 23 and 57) were generated with equal probability from either $N(12, 12 * 0.16)$ or $N(-12, 12 * 0.16)$ (mixture Gaussian); for class "-1", each of these two genes was independently generated from $N(0, 0.16)$. Here, $1_2$ and $I_2$ denote the one-vector and the identity matrix respectively. For those insignificant genes, each was independently drawn from the standard normal $N(0, 1)$. The probabilities for the two classes are equal. The Bayesian classifier with the two significant genes looks like two parallel lines (Fig. 2-4) and the Bayesian bound of the MR is 0.055. For testing, we also independently generated 5000 samples with this underlying generative model. As comparison, we first applied both the linear probit regression method proposed by Lee et al. (2003) and KIGP with an LK (Fig. 2-4-a) to the training set and both of them failed badly to find the correct significant genes, let alone making optimum predictions.

The procedure and all settings of the simulations for this example were identical to those in example 1. We first applied a KIGP with a GK to the training set. The mode of the posterior PDF of the width was found at around 0.81 (Fig. 2-2-c). With GK(0.81) and with a confidence level of 0.95, the cutoff value for NLF was equal to 3.68 (Fig. 2-2-d). Based on the NLF statistic, even in this non-linear setting case, the two prescribed genes were successfully retrieved (Fig. 2-4-c) and its performance (MR = 0.063, Fig. 2-4-d) was very close to the Bayesian bound.
Fig. 2-4: The results of applying KIGP to the non-linear binary simulated example. (a) and (b) are for the LK case; (c) and (d) for the GK case; (e) and (f) for the PK case. All the legends are same as those in Fig. 2-2. (a) The NLF plot for each gene for the LK case; with the cutoff value for NLF, three false positive genes were found significant. (b) The contours of the posterior predictive probability of class “1” for the LK case (assuming the two true significant genes are known). The testing MR is 0.5 (Bayesian bound is 0.055). (c) Same as (a) but for the GK case. (d) Same as (b) but for the GK case and the testing MR is 0.063. (e) Same as (a) but for the PK case. (f) Same as (b) but for the PK case and the testing MR is 0.060.

For the simulation with a PK, after the “kernel parameter fitting phase”, the obtained posterior PMF of the degree, “d=2”, is 0.771, therefore PK(2) was selected. The NLF plot for each gene with PK(2) and the cutoff line for NLF shown in Fig. 2-4-e. The two prescribed genes were discovered correctly with the KIGP. Its performance (MR = 0.060, Fig. 2-4-f) was also very close to the Bayesian bound. Since the performance of KIGP with PK(2) and that of KIGP with GK(1.61) are very close, we can choose either of them and the relatively chosen significant genes to make further predictions.
It is worth highlighting some advantages of using KIGP over most other non-Bayesian approaches. In this example, besides effectively retrieving the significant genes and delivering a performance that is very close to the Bayesian bound in both the linear and the non-linear case, the KIGP further offers a probabilistic prediction for each sample and a posterior distribution for each model parameters. Comparing to a simple binary decision, a probabilistic prediction contains more information and further delivers an uncertainty (or risk) measure of the decision, which will help aid in more reliable decision making. All of these properties mentioned above are valuable for most real applications.

On a side note, one can find the posterior PDF of the width parameter of GK provided very useful information of the fit feature space for the examples in this subsection. Comparing the posterior PDF of the width parameter of these two cases, when the underlying Bayesian classifier can be well approximated by a line, its mode (peak) significantly moves to right (Fig. 2-2-a) of 1; if the Bayesian classifier is very non-linear, its mode moves to the left (Fig. 2-2-c) of 1. The mode of the posterior PDF of the width of GK apparently implies the fitted feature space of a given classification problem. We actually have observed this property of a GK in the illustrative examples given in Chapter 1.

Example 3: Case with a non-linear Bayesian classifier (with multiple significant genes)

In this case, the 10 preset significant genes were generated from a mixture Gaussian distribution with equal probability on $N(1_{10}, I_{10} * 0.1)$ and $N(-I_{10}, I_{10} * 0.1)$ for
the class “1” and from the Gaussian distribution \( N(0_{10}, I_{10} = 0.1) \) for the class “-1”, where \( 0_{10} \) denotes the vector with 10 “0” elements. The probabilities for the two classes were equal. The rest of other insignificant genes were independently generated from the standard normal distribution \( N(0, I) \). The number of testing samples is 5000.

Fig. 2.5: The results of applying KIGP to one of the training sets in example 3. (a) and (b) are for the simulation with a PK; (c) and (d) are for the simulation with an GK. (a) The estimated marginal posterior PMF of the degree parameter \( d \). (b) The NLF plot of each gene for the simulation with the PK(2); the dots mark the prescribed significant genes. For this training set, all 10 preset significant genes and 1 false positive gene were found. (c) The estimated marginal posterior PDF of the width parameter \( r \) (solid line) versus its prior PDF (dotted line). The mode of the posterior PDF of the width is at around 0.64. (d) The NLF plot for each gene for the simulation with the GK(0.64). The legends are same as those in (b). For this training set, all 10 preset significant genes were found with no false positive.

The procedure for this example is same as in example 2. The prior probability for \( \gamma_j = 1 \) was set at 0.01. For both the kernel parameter fitting phase and the gene selection
phase, we ran 20000 iterations and treated the first 10000 as the burn-in period, and for the prediction phase, we ran 5000 iterations and treated the first 1000 as the burn-in period.

In Fig. 2-5, we show the simulation results from applying the KIGP method to the training sets. Figure (a) and (b) are for the simulation with a PK, whereas Figure (c) and (d) are for the simulation with a GK. Based on Fig. 2-5a, PK(2) was chosen after the kernel parameter fitting phase. After the “gene selection phase”, with the yielded cutoff line for the NLF, the KIGP found all 10 prescribed significant genes and one “false positive” gene (Fig. 2-5b). The MR of the testing set was 0.991. In the simulation with a GK, the mode of the posterior PDF for the width was found at around 0.64 (Fig. 2-5c). With the GK(0.64), after the gene selection phase, all 10 prescribed genes were correctly found with no “false positive”. With the found significant genes, we did not find any testing error in the “prediction phase”. Based on the testing MR, we should choose the GK for further analysis. This example not only illustrates the usefulness of the proposed algorithm for problems with a relatively large number of investigated genes, it also further reinforces all the arguments we have made for KIGP framework in Example 2.

2.3.2.2 Examples with A Multi-Classification Model

Two examples were designed in this section to demonstrate the effectiveness of the KIGP framework for a multi-classification microarray analysis problem. In the first example, the Bayesian classifier between each pair of classes with the underlying generative model can be well approximated by a linear function; in the second example, the Bayesian classifiers take a very non-linear form. Similar to the examples shown in the
last section, in both examples, we preset the number of the significant genes as 2, such that we could better graphically display the Bayesian classifiers and the relative performances of the KIGP method. In both examples, the number of training samples is 25, the number of testing samples is 5000; the number of investigated genes is 200; the index of the two prescribed underlying explanatory gene is [23, 57].

The generative models of the gene data in both of the examples are same. With the two significant genes, there are 5 independent clusters, each of which is Gaussian distributed. The covariance matrix for all these 5 clusters is same: \[
\begin{bmatrix}
0.16 & 0 \\
0 & 0.16
\end{bmatrix}.
\]
The mean vectors of the 5 clusters (i.e. cluster 1, cluster 2, cluster 3, cluster 4, cluster 5) are \([0,0]'\), \([1,1]'\), \([-1,1]'\), \([-1,-1]'\), \([1,-1]'\) respectively. The probabilities for each cluster are equal. For those insignificant genes, each of them is independently generated from the standard normal distribution \(N(0,1)\).

The procedure for these two examples is similar to that for the binary-classification simulated examples. Due to the concern on kernel scaling, we did not run the simulation with a PK. In all the simulations in these two examples, the prior probability for \(\gamma_j = 1\) for all \(j\) was set at 0.01; we ran 5000 iterations in both the “kernel parameter fitting phase” (only applicable for the simulations with a GK) and the “gene selection phase” and treated the first 1000 iterations as the burn-in period. In the “prediction phase”, we ran 1000 iterations and treated the first 100 iterations as the burn-in period; the threshold for the “fdr” in the “gene selection phase” was set at 0.05.

We also applied two established kernel-related microarray analysis methods, SVM and PLR (along with the benchmark method PAM), to these simulated examples.
In detail, we ran simulations with either an LK or a GK for a SVM or a PLR. Since RFE is only proper for LK, we only applied UR for SVM/GK and PLR/GK. For the SVM/GK or PLR/GK, since it's extremely difficult to fit kernel width through a LOOCV for a multi-classification problem, we fixed it as the inverse of number of gene used in the model. In each of the simulations, via either RFE or UR, we ranked all genes on a top-down list and decided the optimum number of chosen top genes by running an LOOCV procedure. All the experimental results are summarized in Table 2-1.

<table>
<thead>
<tr>
<th>Method</th>
<th>Ex. 1</th>
<th>Ex. 2</th>
<th>Ex. 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>KIGP/LK</td>
<td>0.106</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>KIGP/GK</td>
<td>0.068</td>
<td>0.06</td>
<td>0.136</td>
</tr>
<tr>
<td>SVM/LK/UR</td>
<td>0.305</td>
<td>F</td>
<td>0.371</td>
</tr>
<tr>
<td>SVM/GK/UR</td>
<td>0.115</td>
<td>0.198</td>
<td>0.175</td>
</tr>
<tr>
<td>SVM/RFE</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>PLR/LK/UR</td>
<td>0.094</td>
<td>F</td>
<td>0.172</td>
</tr>
<tr>
<td>PLR/GK/UR</td>
<td>0.087</td>
<td>0.172</td>
<td>0.175</td>
</tr>
<tr>
<td>PLR/RFE</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>GP_ARD</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>PAM</td>
<td>0.267</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Bayesian</td>
<td>0.047</td>
<td>0.047</td>
<td>0.047</td>
</tr>
</tbody>
</table>

Table 2-1: Performance comparison of different methods for the simulated multi-classification examples. The numbers in the table represent misclassification rates and symbol “F” means “Fail to function”. Ex. 1 and Ex.2 refer to example 1 (the linear example) and example 2 (the non-linear example) in section 2.3.2.2, respectively; Ex. 3 refers to the example in section 2.3.2.3 (the linear example with a mislabeled training sample).
**Example 1: Case with a semi-linear Bayesian classifier**

In this example, we labeled cluster 1 and cluster 4 as class “1”; cluster 3 as class “2”; and cluster 2 and cluster 5 as class “3” (the base class). With this generative model, the Bayesian classifiers between each pair of classes can be well approximated by a linear function with respect to the two prescribed significant genes (Fig. 2-6 b and 2-6 c).

In the simulation with an LK, there is no “kernel parameter fitting phase”. With the preset threshold for the fdr at 0.05, the two prescribed genes were found significant (Fig. 2-6a) after the “gene selection phase”. The contours of the posterior predictive probabilities for each class are plotted in Fig. 2-7a, Fig. 2-7b and Fig. 2-7c respectively. The resulted class decision function is demonstrated by the dotted curves in Fig. 2-6b. The testing MR was 0.106 for this simulation, whereas that of the Bayesian classifier is 0.047. In this example, since the Bayesian classifiers between each pair of classes can all be well approximated by a linear function, the KIGP with an LK still could find the two significant genes and delivered relatively satisfactory performance.

In the simulation with a GK, the posterior PDFs of different width “r” for different classifiers are plotted in Fig. 2-6c, in which the mode for classifier 1 was found at around 0.98 and the mode for classifier 2 was found at around 1.60. After the “kernel parameter fitting phase”, the kernel was fixed as GK(0.98) and GK(1.60) for classifier 1 and 2, respectively. The NLF for each gene was calculated and the two prescribed genes were found significant (Fig. 2-6c) after the “gene selection phase”. The contours of the posterior predictive probabilities for each class are plotted in Fig. 2-7d, Fig. 2-7e and Fig. 2-7f, respectively. The resulted class decision function is shown in Fig. 2-6d. The MR of the independent testing set in this simulation was 0.068, which is very close to that of the
Bayesian classifier (0.047). Since the KIGP with a GK performed significantly better than the KIGP with an LK, we should choose a GK over an LK as the kernel type to do any further analyses.

Fig. 2-6: Results of KIGP of example 1 for a multi-classification model. (a) and (b) for the simulation with an LK; (c), (d) and (e) for the simulation with a GK. (a) and (c) show the NLF plot for each gene, where the dotted line demonstrates the cutoff line with respect to NLF and the asterisks mark the two preset significant genes. In (b) and (d), the resulted class decision function is demonstrated by the dotted curves, where the X-axis and the Y-axis represent the expression values of gene 23 and gene 57, respectively; the asterisks, the circles and the triangles denote the training samples in class "1", class "2" and class "3", respectively; those solid lines present the underlying Bayesian classifiers. The posterior PDFs of different width "r" for different classifiers in the simulation with a GK are plotted in (e).
Fig. 2-7: Contours of the posterior predictive probability for each class in example 1 for the multiclassification model. The X-axis and Y-axis represent the normalized expression values of the two preset significant genes (23 and 57); the numbers associated with contours are the probabilities; the asterisks, the circles and the triangles denote the training samples from class “1”, class “2” and class “3” respectively. (a), (b) and (c) for the simulation with an LK; (d), (e) and (f) for the simulation with a GK.

The results of applying other classifiers to this example are summarized as follows (Table 2-1). With a PAM, we found both prescribed significant genes and a false positive; the testing MR was 0.267. With a PLR, via UR and through a LOOCV procedure, we found the two preset significant genes for this example with either an LK or a GK. The testing MR for a PLR/LK was 0.094, which is very close to that for a KIGP/LK. The testing MR for a PLR/GK was 0.087, which is worse than the KIGP/GK due to the better kernel width fit of the KIGP method. Since in this example, not every class is exclusively linearly separated from all other classes, it is not surprising that the
linear SVM found a false positive gene by adopting the “one against all” strategy. The resulted testing MR was 0.305.; the testing MR with a SVM/GK was 0.115. RFE failed to rank both of the 2 prescribed genes on the top list for the classifier 1 and 3 with either a PLR/LK or a SVM/LK. To our surprise, GP_ARD failed for this case since basically it adopts a kernel function similar to a linear kernel. In the simulation, GP_ARD selected 9 significant genes, but only 1 of them (gene 23) was correct, whereas the rest 8 were “false negative”. GP_ARD also made a “false negative” in missing the gene 57. We suspect that this is possibly due to the limit of the ordinal regression model to a multi-classification problem as we discussed in the introduction chapter. Overall, KIGP/GK delivered the best prediction performance for the testing set because its output class decision function with respect to the 2 significant genes is very close to the Bayesian classifier (Fig. 2-6d). Since the Bayesian classifier between each pair of classes can be well approximated by a linear function, most of other referred linear methods also performed reasonably well (e.g. KIGP/LK, PLR/LK/UR).

**Example 2: Case with a non-linear Bayesian classifier**

In this example, we labeled cluster 2 and 4 as class “1”; cluster 3 as class “2”; and cluster 1 and 5 as class “3”. With this generative model, the Bayesian classifier between class “1” and class “3” takes a very nonlinear form with respect to the two significant genes (Fig. 2-8b and 2-8d).

In the KIGP simulation with an LK, since one of the Bayesian classifiers cannot be approximated by a linear function, KIGP failed to find one of the true significant genes (Fig. 2-8a) and mistakenly picked a wrong gene in the significant gene list. As an
illustration, we show the contour plots of the posterior predictive probabilities for each class given the expression levels of the two preset significant genes in Fig. 2-9a, Fig. 2-9b and Fig. 2-9c. The resulted class decision function as well the Bayesian classifiers are shown in Fig. 2-8b, and the output testing MR was poor (0.293). Clearly, KIGP with an LK did not function well for this example.

Fig. 2-8: Results of example 2 for the multi-classification model. (a) and (b) for the simulation with an LK; (c), (d) and (e) for the simulation with a GK. (a) and (c) show the NLF plot for each gene, where the dotted line demonstrates the cutoff line with respect to NLF and the asterisks mark the two preset significant genes. In (b) and (d), the resulted class decision function is demonstrated by the dotted curves, where the X-axis and the Y-axis represent the expression values of gene 23 and gene 57, respectively; the asterisks, the circles and the triangles denote the training samples in class “1”, class “2” and class “3”, respectively; those solid lines present the underlying Bayesian classifiers. The posterior PDFs of different width “r” for different classifiers in the simulation with a GK are plotted in (e).
Fig. 2-9: Contours of the posterior predictive probability for each class of example 2 for the multiclassification model. (a), (b) and (c) for the simulation with a LK; (d), (e) and (f) for the simulation with a GK. The X-axis and Y-axis represent the normalized expression values of the two preset significant genes (23 and 57); the numbers associated with contours are the probabilities; the asterisks, the circles and the triangles denote the training samples from class “1”, class “2” and class “3” respectively. (a), (b) and (c) for the simulation with an LK; (d), (e) and (f) for the simulation with a GK.

In the simulation with a GK, the posterior PDFs of the width “r” for different classifiers are plotted in Fig. 2-8c. The mode for classifier 1 was found at around 0.51 and the mode for classifier 2 was found at around 1.36. After the “kernel parameter fitting phase”, the kernel was consequently fixed as GK(0.51) for classifier 1 and as GK(1.36) for classifier 2 respectively. The NLF for each gene was calculated after the “gene selection phase” and the two prescribed genes were identified (Fig. 2-8c). The contours of the posterior predictive probabilities for each class with respect to the expression levels of these two genes are plotted in Fig. 2-9d, Fig. 2-9e and Fig. 2-9f.
Fig. 2-10: **KIGP results of the example with a mislabeled training sample.** (a) and (b) for the simulation with an LK; (c), (d) and (e) for the simulation with a GK. (a) and (c) show the NLF plot for each gene, where the dotted line demonstrates the cutoff line with respect to NLF and the asterisks mark the two preset significant genes. In (b) and (d), the resulted class decision function is demonstrated by the dotted curves, where the X-axis and the Y-axis represent the expression values of gene 23 and gene 57, respectively; the asterisks, the circles and the triangles denote the training samples in class “1”, class “2” and class “3”, respectively; those solid lines present the underlying Bayesian classifiers; the mislabeled training sample is marked by both an asterisk and a triangle. The posterior PDFs of different width “r” for different classifiers in the simulation with a GK are plotted in (e).

In the simulation with an LK, due to the non-linearity raised by the mislabeled training sample, the KIGP failed to identify one of the true significant genes (Fig. 2-10a). As an illustration, we show the contour plots of the posterior predictive probabilities for each class even given the normalized expression levels of the two preset significant genes in Fig. 2-11a, Fig. 2-11b and Fig. 2-11c, respectively. The resulting class decision function and the Bayesian classifier are shown in Fig. 2-10b and the testing MR was
0.293, which is dramatically worse than that in its counterpart (example 1 in the last section, 0.106).

![Diagram](image)

Fig. 2-11: Contours of the posterior predictive probability for each class in the example with a mislabeled sample. (a), (b) and (c) for the simulation with a LK; (d), (e) and (f) for the simulation with a GK. The X-axis and Y-axis represent the normalized expression values of the two preset significant genes (23 and 57); the numbers associated with contours are the probabilities; the asterisks, the circles and the triangles denote the training samples from class “1”, class “2” and class “3” respectively, where the mislabeled training sample is marked by both an asterisk and a triangle.

In the simulation with a GK, the posterior PDFs of the width “r” for different classifiers are plotted in Fig. 2-10e. The mode for classifier 1 was found at around 0.99 and the mode for classifier 2 was found at around 1.65. After the “kernel parameter fitting phase”, the kernel functions were fixed as GK(0.99) and GK(1.65) for classifier 1 and classifier 2 respectively. The two prescribed genes were then found significant even with the mislabeled training sample (Fig. 2-10c). The contours of the posterior predictive
probabilities for each class are plotted in Fig. 2-11d, Fig. 2-11e and Fig. 2-11f respectively. The resulting class decision function and the Bayesian classifier are shown in Fig. 2-10d. The MR of the independent testing set for this simulation was 0.136, which is still reasonable and much better than that in the simulation with the KIGP/LK.

We also applied other methods to this example as in example 1 in the last section (Table 2-1). As anticipated, all linear model based methods suffered their performance degradation. PAM, PLR/RFE, SVM/RFE and GPARD failed to function. Via UR, both of the preset significant genes were ranked as the top two genes. Admittedly, due to the L-1 norm cost function characterized in a PLR or SVM, given the right significant gene set, they were less sensitive to mislabeled training samples than an LSSVM or a GP based learner, which essentially are embedded with the L-2 norm cost function. However in this example, with a LOOCV procedure, both SVM/GK/UR and PLR/GK/UR mistakenly picked up the third gene in the ranking list into the prediction model. As a result, both of them performed worse than a KIGP/GK. This example shows that an LOOCV procedure is not robust at all to mislabeled training sample. Furthermore, as we demonstrated in Example 2 in the last section, a UR approach does not necessarily provide a correct significant gene ranking. In this example, KIGP/GK not only selected the correct preset significant genes, but also outperformed the SVM/GK or PLR/GK with UR gene selection.

Interestingly, the modes of the width parameter of the GK for both classifiers in this example (Fig. 2-10e) are very close to those yielded in its counterpart example (Fig. 2-6e). This explains why the KIGP/GK still could perform well while the KIGP/LK (as well as other linear methods) suffered performance degradation in this example.
Comparing to most classic linear methods, KIGP/GK has more freedom of providing a fitted learning function for a given problem. It thus is flexible yet precise and relatively robust to mislabeled training data.

2.3.3 Real Dataset Studies

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Publication</th>
<th>M</th>
<th>p</th>
<th>n</th>
<th>W</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acute Leukemia</td>
<td>Golub et al. (1999) [24]</td>
<td>2</td>
<td>7129</td>
<td>38</td>
<td>34</td>
<td>ALL/AML</td>
</tr>
<tr>
<td>SRBCT</td>
<td>Khan et al. (2001) [34]</td>
<td>2</td>
<td>2308</td>
<td>35</td>
<td>12</td>
<td>EMS/NB</td>
</tr>
<tr>
<td>Colon</td>
<td>Alon et al. (1999) [4]</td>
<td>2</td>
<td>2000</td>
<td>62</td>
<td>0</td>
<td>Tumor/Normal tissue</td>
</tr>
<tr>
<td>Lymphoma</td>
<td>Alizadeh et al. (2000) [3]</td>
<td>3</td>
<td>4026</td>
<td>62</td>
<td>0</td>
<td>Subtypes of lymphoma</td>
</tr>
<tr>
<td>Breast Cancer</td>
<td>Hedenfalk et al. (2001) [29]</td>
<td>3</td>
<td>3226</td>
<td>22</td>
<td>0</td>
<td>BRCA1/BRCA2/sporadic</td>
</tr>
<tr>
<td>Brain Tumor</td>
<td>Pomeroy et al. (2002) [56]</td>
<td>5</td>
<td>5597</td>
<td>42</td>
<td>0</td>
<td>Different tumor types</td>
</tr>
</tbody>
</table>

Table 2-2: Summary of the datasets analyzed in this study. M: Number of class; p: number of investigated genes; n: number of training samples; W: number of testing samples.

Following the similar procedure executed for the simulated examples, KIGPs were applied to six published microarray gene expression datasets: the acute leukemia dataset, the SRBCT dataset, the colon dataset, the lymphoma dataset, the breast cancer dataset and the brain tumor dataset. A brief summary of these datasets is provided in Table 2-2.

In all the experiments presented in this section, the performance of the proposed KIGP was competitive with that of any other referred methods and outstandingly delivered the best performance in a few cases. For each of the datasets, we found several genes that were not reported by the original publication (Table 2-4, 2-6 and Annex A). At the end of this section, we especially address the issue of how to properly measure the
generalized performance without introducing a so called "gene-selection bias", which was neglected by a few publications, including [38] and [83]. The results of the simulations with a GK reveal that linearity is associated with all the studied datasets. As a by-product of the analysis, we also point out a few suspect samples that were possibly mistakenly phenotyped by the original publications, which further testifies that KIGP with a GK is more robust (or less sensitive) to wrongly labeled training samples.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Kernel</th>
<th>LOOCV</th>
<th>Independent Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td># of Errors</td>
<td>APP</td>
</tr>
<tr>
<td>Acute Leukemia</td>
<td>LK</td>
<td>0/38</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>PK</td>
<td>0/38</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>GK</td>
<td>0/38</td>
<td>0.93</td>
</tr>
<tr>
<td>SRBCT</td>
<td>LK</td>
<td>0/35</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>PK</td>
<td>0/35</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>GK</td>
<td>0/35</td>
<td>0.91</td>
</tr>
<tr>
<td>Colon</td>
<td>LK</td>
<td>0/62</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>PK</td>
<td>0/62</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>GK</td>
<td>5/62</td>
<td>0.85</td>
</tr>
</tbody>
</table>

Table 2-3: Performance results for the binary real datasets. "NA" means "not applicable".
### 2.3.3.1 Binary Dataset

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Kernel</th>
<th>Indices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acute Leukemia</td>
<td>LK (20)</td>
<td>4847, 3320, 2020, 5039, 1834, 4499, 1745, 3847, 4196, 1779, 6539, 6376, 3258, 2111, 1882, 1829, 1249, 2121, 2288, 1924</td>
</tr>
<tr>
<td></td>
<td>PK (10)</td>
<td>4847, 3320, 2020, 1834, 3847, 1745, 1247, 6539, 1882, 2001</td>
</tr>
<tr>
<td></td>
<td>GK (8)</td>
<td>3320, 2020, 4847, 461, 5039, 3847, 1834, 4196</td>
</tr>
<tr>
<td>SRBCT</td>
<td>LK (31)</td>
<td>976, 1389, 742, 255, 2050, 481, 823, 545, 246, 2144, 1873, 153, 437, 1601, 1662, 1347, 1110, 1888, 1768, 1764, 575, 1579, 2258, 1770, 29, 1066, 1968, 1327, 2157, 571, 1954</td>
</tr>
<tr>
<td></td>
<td>PK (15)</td>
<td>255, 976, 1389, 742, 2144, 823, 2050, 842, 545, 867, 481, 1662, 1601, 437, 1700</td>
</tr>
<tr>
<td></td>
<td>GK (11)</td>
<td>976, 255, 823, 742, 1389, 2050, 153, 1601, 1888, 481, 422</td>
</tr>
<tr>
<td>Colon</td>
<td>LK (21)</td>
<td>377, 493, 1843, 576, 1772, 14, 1423, 1346, 353, 792, 1635, 1360, 1094, 739, 1549, 175, 1400, 211, 1897, 1210, 1668</td>
</tr>
<tr>
<td></td>
<td>PK (8)</td>
<td>377, 353, 1772, 1400, 1924, 576, 1423, 1843</td>
</tr>
<tr>
<td></td>
<td>GK (8)</td>
<td>377, 493, 249, 267, 245, 765, 513, 14</td>
</tr>
</tbody>
</table>

Table 2-4: Index of the significant genes selected by KIGP for the real binary datasets.

The detailed gene descriptions are provided in Annex A.

Three datasets with only binary-class are analyzed in this section. The performance results for all the three datasets studied in this section are summarized in Table 2-3. The NLF plots for all the KIGP simulations in this section are assembled in Fig. 2-12 and the index of the selected significant genes are listed in Table 2-4. The relative detail descriptions of the identified significant genes are given in Annex A. The
heatmaps of the identified significant genes by the KIGP with the optimum kernel types are provided in Annex B (Fig. A-1 for the leukemia dataset; Fig. A-2 for the SRBCT dataset; Fig. A-3 for the colon dataset). The posterior PDFs of the width parameter for the simulations with a GK are also provided in Fig. 2-13. Through this subsection, we do not consider the kernel type selection problem and shall discuss it later.

Fig. 2-12: NLF plots for the datasets studied in section 2.3.3.1. (a), (b) and (c) for the acute leukemia dataset; (d), (e) and (f) for the SRBCT dataset; (g), (h) and (i) for the colon dataset. In each of the NLF plots, the dotted line shows the cutoff line with respect to NLF and the dots mark the resulting selected significant genes with the kernel in the title.
Fig. 2-13: Estimated posterior PDFs of the width parameter for the simulations with a GK. (a) Acute Leukemia dataset; (b) SRBCT dataset; (C) Colon dataset. The number in each plot marks the mode of the PDF and the dotted lines are the PDFs of the relative priors.

Case 1: Acute leukemia dataset

The acute leukemia dataset was originally published by [24], in which the bone marrow or peripheral blood samples were taken from 72 patients with either acute myeloid leukemia (AML) or acute lymphoblastic leukemia (ALL). The data was divided into two independent sets: a training set and a testing set. The training set consists of 38 samples, of which 27 are ALL and 11 are AML. The testing set consists of 34 samples, of which 20 are ALL and 14 are AML. This dataset contains expression levels for 7129 human genes produced by Affymetrix high-density oligonucleotide micorarrays. The microarray data represent the intensity of gene expression after being rescaled. By using a
weighted voting scheme, [24] made predictions for all the 34 testing samples and 5 of them were reported as being misclassified.

KIGP with a GK, a PK, and an LK was applied to the training set respectively. The prior parameter $\pi_j$ for all $j$ was uniformly set at 0.001. In both “kernel parameter fitting phase” and “gene selection phase”, we ran 30000 iterations and treated the first 15000 iterations as the burn-in period; and in the “prediction phase”, we ran 5000 iterations and treated the first 1000 iterations as the burn-in period.

The NLF plots are shown in Fig. 2-12. For the KIGP simulation with an LK, 20 genes were identified as “significant” at a 0.05 significance level (Table 2-4). Based on this set of genes we then made predictions for the 34 testing samples. We also ran a LOOCV procedure as detailed in section 2.3.1 for the 38 training samples. The performance results are summarized in Table 2-3. This whole procedure was then repeated for the simulation with a PK and with a GK respectively. Specifically, for the “kernel fitting phase” in the simulation with a PK, the resulting posterior probability of the degree parameter $d$ are $\Pr(d=1) = 0.985$ and $\Pr(d=2) = 0.015$. We therefore chose PK(1) as the kernel function for the rest of this simulation. For the “kernel fitting phase” in the simulation with a GK, the mode of the fitting posterior PDF of the width parameter $r$ is around 2.79 (Fig. 2-13a). We therefore chose GK(2.79) as the kernel function for the rest of this simulation.

In the testing phase, KIGP with an LK and a GK both made only 1 error and KIGP/PK made 2 errors for the total 34 testing samples and all of them were perfect for the LOOCV for the training set. We found that many publications (e.g. [38], [80] and [85]) unanimously reported the same error for this testing sample (#31, Fig. A-1) as well.
Only [81] reported zero testing error. However, based on the results of [81], the testing APP was only 0.83, which is much worse than that of a KIGP/LK. We therefore suspect that this misclassified testing sample by KIGP/LK might be phenotyped incorrectly. In order to compare to another established GP related microarray analysis method, GP_ARD ([9]), we also applied GP_ARD to this dataset and 4 testing errors were found, which is very similar to that reported by the original literature [24]. KIGP has an obvious edge over GP_ARD for this dataset.

On a side note, as we have discussed earlier, posterior PDF of width parameter of a GK can disclose some characteristic nature of the proper feature space for a given dataset and problem. Fig. 2-13a illustrates the dominant linearity of this case.

Case 2: Small round blue cell tumor (SRBCT) dataset

The SRBCT data was originally published by [34]. The tumor types include Ewing family of tumors (EWS), rhabdomyosarcoma (RMS), neuroblastoma (NB) and non-Hodgkin lymphoma (NHL). The dataset of the four tumor types is composed of 2308 genes and 63 samples, while 25 blinded testing samples are available. In this study, in order to more easily compare our analysis results to the other established methods such as suggested in [38] and [80], we only focused on two classes, EWS and NB. Thus, there are only 35 training samples (23 EWS and 12 NB) and 12 testing samples (6EWS and 6NB).

We applied the same procedure as we did in the leukemia data case to this dataset. The computational settings were also almost the same except that $\pi_j$ for all $j$ was set at 0.003. The overall performance report is given in Table 2-3. The significant genes found
by KIGP with each kernel type are listed in Table 2-4. The NLF plots are shown in Fig. 2-12.

Specifically, for the "kernel fitting phase" in the simulation with a PK, the resulted posterior probability of $\text{Pr}(d=1)$ is much larger than that for $d=2$. We therefore chose PK(1) as the kernel function for the rest of this simulation. The mode of the fitting posterior PDF of the width parameter $\tau$ for the KIGP/GK simulation is around 2.35 (Fig. 2-13b). We therefore chose GK(2.35) as the kernel function for the rest of this simulation.

In the end, we found that KIGP with a PK outperformed the Artificial Neural Network (ANN) method reported in the original publication ([34]) in terms of APP (0.95 for KIGP/PK versus 0.92 for ANN), although both methods made perfect prediction in the independent testing and the LOOCV. KIGP also noticeably beat GP_ARD ([9]), which made 3 independent testing errors.

**Case 3: Colon dataset**

In this subsection, we show the result when KIGP was applied to the colon dataset ([4]). There are 62 training samples and each of them is labeled by either "Tumor" or "Normal". For each sample, there are 2000 investigated genes.

We first adopted the same procedure as we did for the leukemia data case to this dataset. The computational settings were again almost the same except that $\pi_j$ for all $j$ was set at 0.005. The overall performance report is also given in Table 2-3. The significant genes found by KIGP with each kernel type are listed in Table 2-4. The NLF plots are shown in Fig. 2-12. For the simulation with a PK, the resulted posterior
The probability of $\Pr(d=1)$ is very close to 1. We therefore chose PK(1) as the kernel function for the rest of this simulation. The mode of the fitting posterior PDF of the width parameter $r$ for the KIGP/GK simulation is around 2.38 (Fig. 2-13c). We therefore chose GK(2.38) as the kernel function for the rest of this simulation. There is no testing set available for this case. For the LOOCV, both KIGP/LK and KIGP/PK made 0 errors whereas KIGP/GK made 5 opposite predictions. Comparatively, [9] reported 1 LOOCV error by using 21 significant genes. A more detail discussion of this example is shown in section 2.3.3.3.

2.3.3.2 Multi-Class Datasets

The three multiple-class datasets listed in Table 2-2 are studied in this subsection. The experimental procedure for each dataset is similar to that for the simulated examples in section 2.3.2.2. The overall performance results are listed in Table 2-5 and the associated explanations are described in the following relative context. For each of the KIGP simulations, the normalized-log frequency (NLF) plot is shown in Fig. 2-14 and the index of the selected significant gene set is provided in Table 2-6 and the detail gene description for the breast cancer dataset can be found in Annex A. The heatmaps of the identified significant genes by the KIGP with the optimum kernel types are provided in Annex B (Fig. A-4 for the lymphoma dataset; Fig. A-5 for the breast cancer dataset; Fig. A-6 for the brain tumor dataset). In this section, because of the concern of algorithmic stability due to the kernel scaling problem, we didn’t execute the simulation with a PK as for the binary-class examples.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Kernel Type</th>
<th>LOOCV # of Errors</th>
<th>Test # of Errors</th>
<th>APP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lymphoma</td>
<td>LK</td>
<td>0/62</td>
<td>1.00</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>GK</td>
<td>0/62</td>
<td>0.96</td>
<td>NA</td>
</tr>
<tr>
<td>Breast Cancer</td>
<td>LK</td>
<td>0/22</td>
<td>0.97</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>GK</td>
<td>0/22</td>
<td>0.79</td>
<td>NA</td>
</tr>
<tr>
<td>Brain Tumor</td>
<td>LK</td>
<td>0/42</td>
<td>0.97</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>GK</td>
<td>0/42</td>
<td>0.80</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 2-5: Performance of KIGP for the real multi-class datasets. “NA” means “not applicable”.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lymphoma</td>
<td>LK (9)</td>
</tr>
<tr>
<td></td>
<td>GK (4)</td>
</tr>
<tr>
<td>Breast Cancer</td>
<td>LK (22)</td>
</tr>
<tr>
<td></td>
<td>GK (6)</td>
</tr>
<tr>
<td>Brain Tumor</td>
<td>LK (18)</td>
</tr>
<tr>
<td></td>
<td>GK (22)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Indices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lymphoma</td>
<td>3735, 760, 702, 706, 718, 2805, 3782, 766, 61</td>
</tr>
<tr>
<td></td>
<td>757, 758, 760, 678</td>
</tr>
<tr>
<td></td>
<td>1443, 585, 3009, 1116, 2423, 420</td>
</tr>
<tr>
<td>Brain Tumor</td>
<td>3283, 1263, 5539, 227, 1824, 1367, 297, 1049, 3338, 5213, 2515, 1074, 4563, 2989, 545, 2601, 5124, 4349</td>
</tr>
<tr>
<td></td>
<td>1892, 3283, 1263, 2515, 522, 297, 4210, 4511, 5539, 5208, 1367, 5259, 205, 540, 4349, 2990, 227, 2638, 845, 1218, 570, 2989</td>
</tr>
</tbody>
</table>

Table 2-6: Index of the significant genes selected by KIGP for the real multi-class datasets.
Fig. 2-14: **Interim plots for the examples in subsection 2.3.3.2.** (a), (b) and (c) for the lymphoma dataset; (d), (e) and (f) for the breast cancer dataset; (g), (h) and (i) for the brain tumor dataset. The posterior PDFs of the width parameter “r” for the different classifiers with different GKs are plotted in (c), (f) and (i), in each of which the number marks the mode of the PDF and the dotted line is the PDF of the prior. In each of the NLF plots, the dotted line shows the cutoff line with respect to NLF and the dots mark the resulting selected significant genes with the kernel in the title.

In all the simulations in this subsection, the prior parameter $\pi_j$ for all $j$ was uniformly set at 0.005. In both the “kernel parameter fitting phase” and the “gene selection phase”, we ran 15000 iterations and discarded the first 5000 iterations as the burn-in period; in the “prediction phase”, we ran 5000 iterations and threw away the first 1000 iterations as the burn-in period. In the end of the “kernel fitting phase” for the KIGP/GK simulations, the estimated PDFs for the kernel width for each dataset are plotted in Fig. 2-14. Since there is no testing sample set for any of these datasets, we can only report the predictive measure for the training set based on the LOOCV procedure (Table 2-5).
We also applied other methods, including SVM, PLR GP_ARD and PAM to these three datasets. The procedure and method description is same as those in section 2.3.2.2 and section 2.3.2.3. The performance comparison results are summarized in Table 2-7, from which one can see that in all of the three real data examples, the KIGP method was very effective and consistently delivered the best performance comparing to all other referred methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Lymphoma</th>
<th>Breast Cancer</th>
<th>Brain Tumor</th>
</tr>
</thead>
<tbody>
<tr>
<td>KIGP/LK</td>
<td>0/62 (10)</td>
<td>0/22 (22)</td>
<td>0/42 (18)</td>
</tr>
<tr>
<td>KIGP/GK</td>
<td>0/62 (4)</td>
<td>0/22 (6)</td>
<td>0/42 (22)</td>
</tr>
<tr>
<td>SVM/LK/UR</td>
<td>0/62 (41)</td>
<td>0/22 (50)</td>
<td>1/42 (40)</td>
</tr>
<tr>
<td>SVM/GK/UR</td>
<td>5/62 (17)</td>
<td>0/22 (12)</td>
<td>9/42 (50)</td>
</tr>
<tr>
<td>SVM/RFE</td>
<td>0/62 (15)</td>
<td>0/22 (6)</td>
<td>0/42 (20)</td>
</tr>
<tr>
<td>PLR/LK/UR</td>
<td>0/62 (11)</td>
<td>0/22 (10)</td>
<td>3/42 (22)</td>
</tr>
<tr>
<td>PLR/GK/UR</td>
<td>0/62 (11)</td>
<td>0/22 (10)</td>
<td>3/42 (16)</td>
</tr>
<tr>
<td>PLR/RFE</td>
<td>0/62 (8)</td>
<td>0/22 (6)</td>
<td>0/42 (20)</td>
</tr>
<tr>
<td>GP_ARD</td>
<td>0/62 (25)</td>
<td>7/22 (20)</td>
<td>6/42 (38)</td>
</tr>
<tr>
<td>PAM</td>
<td>1/62 (1987)</td>
<td>0/22 (48)</td>
<td>1/42 (5521)</td>
</tr>
</tbody>
</table>

Table 2-7: Performance comparison of different methods for the real multi-class datasets. The format for each cell of the table is: "# of errors / # of training samples (number of selected genes)".

**Case 1: Lymphoma dataset**

The lymphoma dataset was originally published by [3] to utilize microarray to distinct different subtypes of lymphoma. The microarray contains 4026 investigated genes and all the 62 available samples belong to 3 classes.
We applied the KIGP framework to this dataset and all the results are found in Table 2-5. In the simulation with a GK specifically, the modes of the posterior PDFs of the width parameter were found at around 3.45 and 3.13 for classifier 1 and classifier 2 respectively (Fig. 2-14c), which implies that the Bayesian classifier between any two classes of this dataset can all be approximated by a linear function. Therefore, we expect that a linear method would work well for this example.

In the end of the "gene selection phase", we identified the significant gene set (Table 2-6) and continued the "prediction phase" and we did not find any LOOCV errors in each of the KIGP simulations (Table 2-5). As suggested by Occam's razor, if two given models can both well explain (or predict) the data, one always should use the simpler one as the complicated model will be penalized by a stronger Occam's factor ([44]). On this regard, we noticed that KIGP/GK made no error by only using 4 genes (Table 2-7), which is significantly better than any other methods listed therein.

Case 2: Breast cancer dataset

The hereditary breast cancer data used in this example was published by [29], in which DNA microarrays were used in conjunction with classification algorithms to show the feasibility of using the differences in global gene expression profiles to separate BRCA1 and BRCA2/sporadic. 22 breast cancer tumors were examined: 7 with BRCA1 (class 1), 8 with BRCA2 (class 2) and 7 considered sporadic (base class). 3226 genes were investigated for each sample.
With the similar procedure applied for the lymphoma dataset, we obtained performance results for each of the KIGP simulations (Table 2-5). The indices of the identified significant gene sets are in Table 2-6. The modes of the posterior PDFs of the width parameter in the KIGP simulation with a GK were found at around 3.69 and 3.97 for classifier 1 and classifier 2 respectively (Fig. 2-14f), which again implies that the Bayesian classifier between any two classes for this dataset can be precisely approximated by a linear function.

With the identified significant gene sets (Table 2-6) we again made no error in the LOOCV test in each of the KIGP simulations (Table 2-5). In particular, KIGP/GK achieved an errorless prediction only by only using 6 genes. For other referred methods, we found that either SVM/RFE or PLR/RFE with a linear kernel also made perfect prediction by using only 6 genes (Table 2-7). KIGP nonetheless still delivered the best performance (tied) for this trivial dataset.

**Case 3: Brain tumor dataset**

The brain tumor dataset was first published in [56]. In this dataset, there are 5 different brain tumor types for 42 samples. Each sample contains expression levels of 5597 genes. There is no blind independent testing set available. Since this dataset has the largest number of candidate classes among all the 6 real datasets, it’s a good example to testify the effectiveness of the multi-classification KIGP framework for a problem with a large number of classes.
The same procedure adopted in the first 2 cases was applied for the brain tumor dataset. And again there is no LOOCV error found in either of the KIGP simulations (Table 2-5) with the selected gene set (Table 2-6). For the simulation with a GK, the modes of the posterior PDFs of the width parameter were found at around 2.24, 2.34, 2.51 and 2.33 for classifiers 1, 2, 3 and 4 respectively (Fig. 2-14i), implying the unequivocal linearity of this dataset. Comparing to other methods, KIGP/LK used 18 genes to make a prefect prediction, which is very close to the best performer (SVM/RFE, PLR/RFE, Table 2-7).

2.3.3.3 Discussion on The generalized Performance

In most real applications, there is no independent testing set available or testing sample size is not large enough to preclude the possibly noticeable testing bias. Therefore it is important to discuss how to properly measure the generalized performance for a given microarray learner when there is no training set available. Hereby we shall adopt the 3-fold CV procedure as detailed in subsection 2.3.1. Since among our 6 datasets, only the leukemia dataset and the SRBCT data offer a blind testing set, we shall first revisit these two cases.

For each of these two datasets, we first randomly split it into three sets following the procedure direction described in section 2.3.1 for the 3-fold CV. Then we applied KIGP (including all three phases of the learning procedure) to each of the sets, resulting with the values in the two columns entitled with “3-fold CV” in Table 2-8. Obviously, we can see that for both of these two datasets the two predictive fit measures were very consistent for the training set and the testing set. Especially, the APP measures appear
more informative and precise as it indicates generalized performance of a learner with an uncertainty measure. With a multiple 3-fold CV simulation, the average of the "# of Errors" measure, misclassification rate (MR), will clearly give a more consistent evaluation for the generalized performance of a target learner (with less possible split bias).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Kernel Type</th>
<th>3-fold CV</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td># of Errors</td>
<td>APP</td>
</tr>
<tr>
<td>Leukemia</td>
<td>LK</td>
<td>1/38</td>
<td>0.88</td>
</tr>
<tr>
<td></td>
<td>PK</td>
<td>2/38</td>
<td>0.84</td>
</tr>
<tr>
<td></td>
<td>GK</td>
<td>1/38</td>
<td>0.82</td>
</tr>
<tr>
<td>SRBCT</td>
<td>LK</td>
<td>0/35</td>
<td>0.82</td>
</tr>
<tr>
<td></td>
<td>PK</td>
<td>0/35</td>
<td>0.88</td>
</tr>
<tr>
<td></td>
<td>GK</td>
<td>2/35</td>
<td>0.79</td>
</tr>
</tbody>
</table>

Table 2-8: Performance comparison between the 3-fold CV procedure and the independent testing for the leukemia dataset and the SRBCT dataset.

By using this rigorous 3-fold CV procedure as the key measure for the generalized performance for a learner, we shall compare the performance of a KIGP to other state-of-the-art microarray learning methods designed for a classification problem. We noticed that [13] has reported the performance of many popular methods applied to several published datasets. Some of the datasets therein are obviously trivial (or say very learnable with a linear method) based on their results. Interestingly, one of the non-trivial datasets in their article is the colon dataset. Admittedly, [13] did a pre-filtering of the genes based on the Wilcoxon test statistic and ran all the simulations only within a 200-
gene pool, whereas a KIGP simulation in this chapter involve all investigated genes in each of the CVs. However based on the reported simulation procedure, [13] applied the multiple rigorous 3-fold CV described in this chapter and 200-gene pool is large enough to preclude most gene selection bias. Hence, it should be a consistent platform to compare different microarray analysis models (including KIGP) for this dataset. We therefore shall choose this dataset and intensively apply the multiple 3-fold CV procedure to it. The average MR measure for the multiple KIGP 3-fold CV simulations with each candidate kernel type are listed in Table 2-9. For KIGP with (PK, GK, LK), we took 5 independent rigorous 3-fold CVs to 62 samples and reported the average MR. The algorithm set for the procedure for each KIGP simulation is similar to the colon dataset simulations in section 2.3.3.2. For most of the remaining classifiers in Table 2-9, the results were reported in [13] and the simulation procedures can be found therein. For the random forest (RF) method, we adopted the result for this dataset reported in [14], in which a version of random forest method specially designed for microarray analysis is developed. One can see that, KIGP with a GK worked very close to the best classifier shown in the table (PAM and DLDA).

<table>
<thead>
<tr>
<th>Method</th>
<th>KIGP /LK</th>
<th>KIGP /PK</th>
<th>KIGP /GK</th>
<th>BagBoost</th>
<th>Boosting</th>
<th>RF</th>
<th>SVM</th>
<th>GPARD</th>
<th>PAM</th>
<th>DLDA</th>
<th>kNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR</td>
<td>0.198</td>
<td>0.161</td>
<td>0.129</td>
<td>0.161</td>
<td>0.191</td>
<td>0.159</td>
<td>0.151</td>
<td>0.201</td>
<td>0.119</td>
<td>0.129</td>
<td>0.164</td>
</tr>
</tbody>
</table>


92
Based on the MR of the 3-fold CV, obviously KIGP/GK has better generalized performance than KIGP/LK or KIGP/PK. If we check the results given in section 2.3.3.1 for the colon dataset, an interesting finding is that, based on the results of the LOOCV for the prediction phase, KIGP/GK made 5 testing errors. However, since KIGP/GK performed significantly better than KIGP/LK in the rigorous CV test, we trust more of the analysis results yielded in the KIGP/GK simulation. The description of the significant genes found by KIGP/GK is provided in Annex A. When we checked the heat map of these genes (Fig. A-3), we found that a few training samples, particularly including samples #18, #20, #45, #49 and #56, are different from other samples in their same labeled class through this set of genes. However, they are very consistent with those samples in their opposite class. In fact, these samples were almost always misclassified by KIGP/GK in the multiple rigorous 3-fold CV tests. We therefore suspect that these samples are mistakenly phenotyped. We believe that, if this is the true statement for the colon dataset, it is probably can well explain why none of any other learning methods referred in Table 2-9 performed well for this colon dataset. This also supports the nature of a KIGP/GK being less sensitive to the mislabeled training samples than a KIGP/LK, as illustrated in the simulated example in section 2.3.2.3.
Chapter 3

Building Kernels from Microarray:

A Natural Kernel Approach

As discussed in section 1.2, kernel function, as core of any kernel-based learning algorithm, uniquely determines properties of the induced feature space for a given problem. Therefore, how to build an appropriate kernel for a given application problem has always been one of the fundamental elements of this class of learning methods. In theory, as [40, 41] stated, with a properly chosen/built feature space and smoothing parameter, a kernel SVM can approximately achieve the Bayesian bound with sufficient training samples. The methods for building and/or selecting a proper kernel have been one of the most active research topics in kernel-based learning field. Specially, the optimality of such kind of problems is varied for different specific applications, which makes this problem even more challenging.

In Chapter 2, when choosing an LK, a GK or a PK, we implicitly assume that significant genes are independent of each other. But as many researchers have suggested, for a target disease, usually functional significant genes are more often formed in several clusters ([3, 4, 29, 53]). Hence, building a kernel while considering this feature is a very interesting approach for improving the overall performance of a KIGP. This issue has not
been studied before and we address it in this chapter. Roughly speaking, we propose an algorithm to build kernel function for a microarray multi-classification problem via using the concepts of natural kernel.

Throughout the rest of this chapter, we first introduce the general concepts of natural kernel; then we thoroughly present the natural kernel building procedure under a multi-classification KIGP framework, which is specifically termed as “natural kernel-imbedded Gaussian Process” (NKIGP); finally, we shall show a few examples to demonstrate the correctness and effectiveness of the method of NKIGP.

3.1 Basic Concepts of Natural kernel

In general, the choice of a kernel function for a kernel-based learning method should reflect our prior knowledge of the solution for the target problem. The derivation for a KIGP framework given in section 1.2.4 demonstrates this mathematical implication in a Bayesian inference manner. That is, we introduce the kernel concept into the KIGP framework by specifying the prior properties of the correlation between different samples, which is embodied by a kernel function.

Other than directly adopting a pre-selected regular kernel function such as in Chapter 2, some researchers have suggested several kernel-building techniques via using target feature as orientation to solve a specific problem. For example, [42] introduced a “string subsequence kernel” to deal with a text categorization problem. By developing a so-called “locality improved kernel”, [65] reported favorable results for an image processing problem. In our study, we shall focus on an established approach for building a kernel function, “natural kernel”. Particularly, our proposed kernel-building algorithm
is a derivative of "Fisher kernel" ([63]), which is a specific form of the natural kernel class.

In principle, a natural kernel (NK) is defined over a structured observation object of arbitrary size. A generative model should be presumed for the object. This generative model represents our prior knowledge for the target objects and will ultimately determine the overall performance of the developed learner. The basic concepts of natural kernel were first brought by [32, 33] to exploit the possible way to predict structural and functional features of a protein based on its amino-acid sequence via using a discriminative leaning method, SVM. They assumed that a hidden Markov model (HMM) is a proper generative model for amino-acid sequence of a protein. Then they successfully introduced an efficient way to merge this generative approach into the discriminative SVM model and satisfactory results were reported. Later, for a binary classification problem, [71] further extended the original "class-independent" algorithm suggested in [32, 33] to a "class-conditional" version of building a natural kernel for a binary classification problem. The method is named by "tangent of posterior odds (TOP)" kernel. In the rest of this subsection, we shall first briefly go over the concepts of building a natural kernel. Afterward we shall present our approach to a multi-classification microarray analysis problem.

To formulate the method of building a natural kernel, we need to first introduce some basic concepts of information geometry. Consider a family of generative models for the data (or more precisely, object) $x$. We assume the likelihood distribution is denoted by $P(x | \theta)$ based on a generative model. Without losing generality, we further assume $P(x | \theta)$ is smoothly parameterized by the model parameter set $\theta = [\theta_1, \theta_2, ..., \theta_J]$, forming
a statistical manifold in the space of all probability functions. The key idea for building a
natural kernel is to find an induced metric for the training object \( x \) based on its geometric
structure. Often, rather than dealing with \( P(x|\Theta) \) directly, one instead needs to focus on
its logarithm, \( L(\Theta) = \log(P(x|\Theta)) \).

The key for building a natural kernel function is to calculate the so-termed "score
vector" of \( P(x|\Theta) \), which is defined by

\[
\phi_\theta(x) = \frac{\partial (\log(P(x|\Theta)))}{\partial \theta} \tag{3-1}
\]

Obviously, the magnitude of the components of the score vector \( \phi_\theta(x) \) specifies the
extent to which a change in a particular model parameter of the set \( \Theta \) could affect the
probability of generating the object \( x \). The relationship between the score vector to
sufficient statistics is throughly discussed in [31].

Since the manifold of \( \log(P(x|\Theta)) \) is Riemannian, there is a metric defined on
the space of the scores, with metric tensor given by the inverse of the relevant Fisher
information matrix. The Fisher information matrix of \( \log(P(x|\Theta)) \) is formulated by:

\[
I = E_p[\phi_\theta(x)\phi_\theta(x)^T] \text{, or for each element of matrix } I,
\]

\[
[I]_{ij} = E_p[\partial_{\theta_i}(\log(P(x|\Theta)))\partial_{\theta_j}(\log(P(x|\Theta)))] \text{ for } i, j = 1, 2, \ldots, J \tag{3-2}
\]

In (3-2), operator \( E_p[.] \) denotes the expectation with respect to the density function
\( P(x|\Theta) \). With the score vector and its associated Fisher information matrix as defined in
(3-1) and (3-2), respectively, one can calculate the value of a Fisher kernel (FK) function
with respect to two observation objects \( x \) and \( x' \) by

\[
K(x, x') = \phi_\theta(x)^T I^{-1} \phi_\theta(x') \tag{3-3a}
\]
Given a generative model, there are two key steps to build a Fisher kernel function. The first step is to make an appropriate estimation for the parameter set of the generative model, $\theta$, based on the training data. For instance, in [28], the authors used an expectation-maximization (EM) algorithm to retrieve the fitted HMM model parameters for an amino-acid sequence. The other key step is to calculate the Fisher information matrix. In many real applications however, it might be very difficult or even infeasible to obtain a closed form for the Fisher matrix. Therefore properly simplifying the generative model usually is a critical part of building a functional Fisher kernel function. Another alternative to avoid the Fisher matrix calculation in (3-3a) is to simply replace it by an identity matrix, which yields a plain kernel:

$$K(x,x') = \varphi_\theta(x)^T \varphi_\theta(x')$$  \hspace{1cm} (3-3b)

For some application, this simplified version of a Fisher kernel might not decrease much of the overall performance of the built learner. We however always recommend using the full version of the Fisher kernel function defined in (3-3a), as long as a closed form of the Fisher matrix is available. The reason to do so is briefly described in the following paragraph.

First of all, a Fisher information matrix by its nature would whiten its associated score vector. As the result, eigenvalues of a Fisher kernel operator are all equal to 1 (isotropic) in the feature space. In other words, arc length on the manifold of the statistical space spanned by $P(x \mid \theta)$ is invariant to re-parameterizations of the set $\theta$. On the other hand, based on the basic kernel-induced learning theory, it is always preferable to use a kernel, such that eigenvalues of its associated operator are all equal, because this
would maximize the capacity of a linear class of functions in feature space. The detailed discussions of this learning theory are well contained in [10] or [63].

Rather than Fisher kernel and plain kernel, there are also some other forms of a natural kernel function. For example, in [32, 33], the authors suggest using the following formula to build a natural kernel function with respect to two objects in the observation space:

$$K(x, x') = \exp \left( -\frac{\|\phi(x) - \phi(x')\|^2}{2r^2} \right)$$  \hspace{1cm} (3-3c)

In (3-3c), the definitions of the L-2 norm operator and the kernel width \( r \) are the same as those defined for a GK (Eq. 2-1c). In order to distinguish the three different natural kernel forms defined in (3-3), throughout this study, we shall refer to the kernel in (3-3a) as natural Fisher Kernel (NFK); the kernel in (3-3b) as natural plain kernel (NPK); and the kernel in (3-3c) as natural Gaussian kernel (NGK).

Obviously, the natural kernels defined in (3-3) are not related to class label. With a classification problem, a set of class label for the training set is available. Therefore it should yield better performance if the information of the training label can be considered when building a natural kernel. Based on the fundamentals used by [32, 33], for a binary classification problem, [71] proposed a “class-conditional” version of natural kernel, tangent of posterior odds kernel, to which we refer as natural TOP kernel (NTOPK). Instead of focusing on log-likelihood, feature score vector in an NTOPK is obtained by taking derivative of the posterior log-odds between the two classes of the given generative model with respect to each of the model parameters. The resulted kernel
function is fairly similar to a plain kernel (Eq. 3-3b) except that it takes the training class label under consideration as well. The details of building an NTOPK are found in [71].

3.2 Natural Kernel Applied to a Microarray Classification Problem

In this section, following the principles detailed in section 3.1, we present a natural kernel building approach for a microarray multi-classification problem. We shall also adopt the KIGP framework introduced in Chapter 2 and merge this kernel building technique into it.

Since the multi-classification KIGP framework introduced in section 2.2 is a seamless generalized form of the binary version introduced in section 2.1, in this section we only consider a multi-classification problem and the model is thus the same as that described in section 2.2.

In order to extend the original “class-independent” version of the NK building method, we first propose a class-conditional model. In detail, we assume that data under each class follows a distinct generative model. Therefore, under each class, we can build a kernel function following the general guideline through Eq. (3-1), (3-2) and (3-3). In the end, we calculate the final kernel function by taking the expectation of the kernel function conditional on each class with respect to the probability of the class label. We illustrate the details of this approach in what follows to build a natural kernel for a multi-classification microarray analysis problem with some presumed generative models.
3.2.1 Natural Gaussian Fisher Kernel (NGFK)

Our first proposed natural kernel is based on the following assumption. For all samples under a distinct class, the microarray data of the significant genes can be well modeled by a multivariate Gaussian distribution. For example, [53] adopted this assumption for retrieving missing data in a microarray. Mixture multivariate Gaussian basically is a natural and appropriate assumption for most applications, although sometimes it might underestimate the possibly flat tail of the underlying true distribution for some of the significant microarray data. We shall address this issue in the next section.

With a multivariate Gaussian distribution, if the dimension (or in other words, the number of significant genes) is too high, there would be too many model parameters, most of which are those non-diagonal elements of the covariance matrix. Since the number of available training samples is always limited in a typical microarray analysis problem, we should always avoid building a learning model that has too many parameters. In this study, our solution to this problem is to project the significant gene data onto an orthogonal space via an eigenvalue decomposition procedure. More formally, let \( X_{T,y} \) denote the selected training gene data (with respect to a given gene selection vector \( \gamma \) ) for the training samples in class \( y \). Matrix \( X_{T,y} \) takes the same format used in Eq. (1-4) and we use \( n_y \) to represent the number of training samples in class \( y \) (apparently \( n = \sum_y n_y \)). Thus,

\[
X_{y,y} = \begin{bmatrix} X_{11,y}, X_{12,y}, \ldots, X_{1q,y} \\ \vdots \\ X_{n_1,y}, X_{n_2,y}, \ldots, X_{n_q,y} \end{bmatrix} = \begin{bmatrix} x_{y1,y} \\ \vdots \\ x_{ym,y} \end{bmatrix} \tag{3-4}
\]
Under the assumption in this subsection, we have $x_{\pi_i}$ for $i = 1, \ldots, n_y$ is of a multivariate Gaussian distribution. That is $x_{\pi_i} \sim MN(\mu_y, \Sigma_y)$ (MN is the abbreviation for a multivariate normal/Gaussian distribution). In this study, we shall use the sample moments to estimate the two parameter sets. That is,

$$m_y = \frac{1}{n_y} \sum_{i=1}^{n_y} x_{\pi_i}$$

$$S_y = \frac{1}{n_y - 1} (X_{\pi_i,y} - m_y 1_{n_y}) (X_{\pi_i,y} - m_y 1_{n_y})'$$

(3-5)

In (3-5), $1_{n_y}$ denotes an $n_y$ by 1 vector with all elements equal to 1. The sample mean and variance matrix pair yielded in (3-5), $(m_y, S_y)$, is an unbiased estimation for the population mean and variance matrix pair $(\mu_y, \Sigma_y)$. Obtaining the sample variance matrix $S_y$, we then apply an eigenvalue decomposition (CD, [6, 59]) to it, yielding

$$S_y = T_y \Lambda_y T_y'$$

(3-6)

In (3-6), $\Lambda_y$ is a diagonal matrix containing the eigenvalues and $T_y$ is a univariate matrix. Without losing generality, we assume the diagonal elements of the matrix $\Lambda_y$ are arranged in a descending order. Under this setting, obviously, the first column of $T_y$ is the first principal component (PC) of matrix $X_{\pi_i,y}$ and the first diagonal value of the matrix $\Lambda_y$ is its associated eigenvalue. And so on so forth for the remaining columns of $T_y$.

It is straightforward to show that the sample mean and the sample covariance matrix of the projected data matrix $X_{\pi_i,y}T_y$ are $T_y m_y$ and $\Lambda_y$, respectively. That is,
components of the projected data are orthogonal to each other. We thus define the natural kernel function (given a class) over the projected data. Because usually the number of selected significant genes, \( q \), is larger than the number of samples for each class, it is beneficial to eliminate those components with very small eigenvalues. Throughout this study, we shall only choose the column of matrix \( T_y \), whose associated eigenvalue is larger than 1% of the sum of the all eigenvalues in matrix \( \Lambda_y \). We denote this new (after the elimination step) linear operator by \( T_{0y} \). If we assume that the number of the kept eigenvalues of \( \Lambda_y \) under a given threshold (such as 1%) is \( g_y \), then \( T_{0y} \) is the first \( g_y \) columns of matrix \( T_y \). More formally,

\[
T_{0y} = T_y [I_g ; 0] \quad (3-7a)
\]

In (3-7a), \( I_g \) is an identity matrix with dimension \( g_y \); \( 0 \) denotes a zero matrix with dimension \( q - g_y \) by \( g_y \). Obviously, \( T_{0y} \) is a \( q \) by \( g_y \) matrix and it functions as an operator to project a vector in a space with dimension \( q \) onto a new space with lower dimension \( g_y \).

Given the data matrix \( X_{t,y} \) under class \( y \), following the procedure given in (3-5), (3-6) and (3-7a), we obtain a linear operator \( T_{0y} \). The sample mean and the sample variance matrix for the projected data, \( \bar{X}_{t,y} = X_{t,y} T_{0y} \), are obviously \( T_{0y} \bar{m}_y \) and \( \Lambda_{0y} \) respectively. \( \Lambda_{0y} \) is a diagonal matrix and its elements are the first \( g_y \) elements of \( \Lambda_y \).

With the same notations adopted in Eq. (3-7a), it is given by,

\[
\Lambda_{0y} = [I_g ; 0]' \Lambda_y [I_g ; 0] = diag(\lambda_1, \lambda_2, ..., \lambda_{g_y}) \quad (3-7b)
\]

We thereby obtain a generative model for microarray data under each class.
With a testing sample $x$, we denote the vector of significant test gene data by $x_T$. Then with the operator yielded in (3-7), we project it onto an orthogonal space under class $y$, obtaining the conditional projected data $x_T | y = x_T^T \mathbf{T}_{Gy} = [\tilde{x}_{y,1}, \tilde{x}_{y,2}, ..., \tilde{x}_{y,g}]'$. If this sample belongs to class $y$, it should be of a multivariate Gaussian distribution with mean $\tilde{m}_{Gy} = \mathbf{T}_{Gy} \mathbf{m}_y = [m_{Gy,1}, m_{Gy,2}, ..., m_{Gy,g}]'$ and covariance matrix $\Lambda_{Gy}$. Therefore, the logarithm of the probability density function of $x_T | y$ is

$$
\log(P(x_T | y, \tilde{m}_{Gy}, \Lambda_{Gy})) = \sum_{j=1}^{g} \log(P(\tilde{x}_{y,j} | y, \tilde{m}_{Gy,j}, \lambda_j))
$$

where $\tilde{x}_{y,j} | y, \tilde{m}_{Gy,j}, \lambda_j \sim \text{Normal}(\tilde{m}_{Gy,j}, \lambda_j)$ for $j = 1, ..., g$.

Differentiating the left hand side of (3-8) with respect to each of the model parameters, we get the score vector:

$$
\phi(x) | y, \tilde{m}_{Gy}, \Lambda_{Gy} = [\varphi_{\mu,1}, ..., \varphi_{\mu,g}, \varphi_{\lambda,1}, ..., \varphi_{\lambda,g}]', \text{ where for } j = 1, ..., g
$$

$$
\varphi_{\mu,j}(x) | y, \tilde{m}_{Gy}, \Lambda_{Gy} = (\tilde{x}_{y,j} - \tilde{m}_{Gy,j}) / \lambda_j,
$$

$$
\varphi_{\lambda,j}(x) | y, \tilde{m}_{Gy}, \Lambda_{Gy} = -((\tilde{x}_{y,j} - \tilde{m}_{Gy,j})^2 / 2 + 1 / (2\lambda_j))
$$

The analytical formula for the associate Fisher information matrix for the score vector provided in (3-9), $\mathbf{I}_{Gy}$, is available:

$$
\mathbf{I}_{Gy} | y, \tilde{m}_{Gy}, \Lambda_{Gy} = \text{diag}(I_{\mu,1}, ..., I_{\mu,g}, I_{\lambda,1}, ..., I_{\lambda,g}), \text{ where for } j = 1, ..., g
$$

$$
I_{\mu,j} | y, \tilde{m}_{Gy}, \Lambda_{Gy} = 1 / \lambda_j,
$$

$$
I_{\lambda,j} | y, \tilde{m}_{Gy}, \Lambda_{Gy} = \lambda_j^2 / 2
$$

Note that, the Fisher information matrix only depends on the training set.
In practice, based on a training set, we can first calculate the projecting matrix $T_{G_y}$, the estimated parameters $\hat{m}_{G_y}$ and $\hat{\Lambda}_{G_y}$, and the Fisher matrix $I_{G_y}$ under each candidate class. Then, for any two vectors $x_i$ and $x_i'$, under each class $y$, we project them to an orthogonal space via using $T_{G_y}$ as the operator, obtaining $\hat{x}_i$ and $\hat{x}_i'$. After that, we apply the procedure described in Eq. (3-9) to get the conditional score vector $\varphi(\hat{x}_i | y)$ and $\varphi(\hat{x}_i' | y)$. The resulted conditional kernel function is thereby:

$$K(x,x'|y) = \varphi(\hat{x}_i | y)^T I_{G_y}^{-1} \varphi(\hat{x}_i' | y)$$  \hspace{1cm} (3-11a)

After applying the same procedure under each class, the overall kernel function is defined as the expectation with respect to the probability of each class. That is

$$K(x,x') = \sum_{y=1}^{M} K(x,x'|y) P(y)$$  \hspace{1cm} (3-11b)

In (3-11b), the probability of class $y$, $P(y)$, is estimated by

$$P(y) = n_y / n, \text{ for } y = 1, 2, ..., M$$  \hspace{1cm} (3-11c)

Through the procedure described above, one is able to build a natural Fisher kernel based on a class-conditional multivariate Gaussian distribution model for the significant gene data. In the rest of this study, we shall refer to this kernel as natural Gaussian Fisher kernel (NGFK). We summarize the NGFK building procedure in the following algorithm:

Assuming the training set is $\{(x_{ij}, y_i) | i = 1, ..., n\}$, where $y_i \in \{1, ..., M\}$; and the testing set is $\{(\hat{x}_{ij}) | i = 1, ..., W\}$. First, estimate $P(y)$ through (3-11c). Then, initialize the kernel matrix $K$ with an $n$ by $n$ zero matrix; initialize the testing kernel vector set $k$ with a $W$ by $n$ zero matrix; initialize the testing variance
vector \( \mathbf{k} \) with a \( W \times 1 \) zeros vector. Under each class \( y \) (\( y \) from 1 to \( M \)), we execute the following steps:

1. Selecting all the training samples with label \( y \) and forming them as the new matrix \( \mathbf{X}_{y,y} \).

2. Calculating the sample mean \( \mathbf{m}_y \) and the sample variance matrix \( \mathbf{S}_y \) via (3-5), followed by applying an eigenvalue decomposition to \( \mathbf{S}_y \) (Eq. 3-6) to get the eigenvector matrix \( \mathbf{T}_y \) and the eigenvalue diagonal matrix \( \mathbf{A}_y \) (diagonal elements are in a descending order).

3. Eliminating the trivial eigenvector (with eigenvalue that is smaller than a small fraction, say 1\%, of the sum of all eigenvalues) to form the projecting matrix \( \mathbf{T}_{Gy} \) and the associated eigenvalue matrix \( \mathbf{A}_{Gy} \) via (3-7).

4. Calculating the mean of the projected data by \( \bar{\mathbf{m}}_{Gy} = \mathbf{T}_{Gy} \mathbf{m}_y \).

5. Calculating the Fisher information matrix \( \mathbf{I}_{Gy} \) via (3-10).

6. Multiplying \( \mathbf{T}_{Gy} \) to both the training and the testing set, obtaining their projected datasets \( \mathbf{X}_{y,y} \mathbf{T}_{Gy} \) and \( \tilde{\mathbf{X}}_{y,y} \mathbf{T}_{Gy} \), respectively.

7. Row (each sample) by row with the projected training dataset, calculating its fisher score via (3-9). Overall, generating a score matrix for the training set \( \mathbf{\Phi}_{Gy} \), each column of which contains the score vector of a training sample.

8. For each row (each sample) of the projected testing dataset, also calculating its fisher score via (3-9), generating a score matrix for the testing set \( \tilde{\mathbf{\Phi}}_{Gy} \), each column of which contains the score vector of a testing sample.
9. \( K \leftarrow K + [\Phi_{Gy}^T \Sigma_{Gy}^{-1} \Phi_{Gy}] p(y) \);

10. \( k \leftarrow k + [\tilde{\Phi}_{Gy}^T \Sigma_{Gy}^{-1} \tilde{\Phi}_{Gy}] p(y) \);

11. \( \kappa_i \leftarrow \kappa_i + [\tilde{\phi}_{Gy}(\cdot,i)^T \Sigma_{Gy}^{-1} \tilde{\phi}_{Gy}(\cdot,i)] p(y) \) for \( i = 1, \ldots, W \).

(3-12)

With algorithm (3-12), all necessary kernel matrices (\( K \), \( k \) and \( \kappa \)) in order to fulfill a kernel-related supervised learning method are calculated. In step 11 of (3-12), \( \tilde{\phi}_{Gy}(\cdot,i) \) represents the \( i \)-th column of the matrix \( \tilde{\Phi}_{Gy} \). In step 2 of (3-12), since the sample covariance matrix \( S_y \) is symmetric and non-negative definite, there are fast and robust algorithms to do the eigenvalue decomposition, such as Cholesky decomposition. The most computations in (3-12) happen in the eigenvalue decomposition step 2 with a complexity scale \( O(q^3) \), where \( q \) is the number of significant genes selected in the learning model. In most kernel-related microarray analysis method with a gene selection strategy (such as KIGP) this complexity will not be a concern since \( q \) is always relatively small. Actually \( O(q^3) \) can be fairly close to \( O(n^3) \) for a KIGP. Nevertheless, building a NGFK usually still takes more time (\( O(n^3) \)) than calculating a regular kernel matrix (\( O(n) \) to \( O(n^2) \)).

3.2.2 Natural Student-t Fisher kernel (NTFK)

In many real microarray applications, outlier sample (or outlier gene expression data in a sample) is not a rarity. Under this kind of scenario, a Gaussian distribution model may underestimate the probability of occurrence of extreme data. Therefore, a
distribution model with flatter tail, such as a Student-t distribution is worth considering. Multivariate Student-t distribution is obviously a natural extension of a multivariate Gaussian distribution, it is however extremely difficult to find the close analytical form for the Fisher information matrix even for uncorrelated projected data. There are two alternatives for solving this problem. The first one is to only build a plain kernel so that we can avoid the computation of the Fisher information matrix. The other one is to simplify the generative distribution model. We hereby shall adopt the second approach.

We still only focus on the data after the orthogonal projection as we do with a multivariate Gaussian model in the last subsection. The basic assumption here is that, under class $y$, we assume that components of the projected data are independent with each other and of a same univariate Student-t distribution with degree $\nu$. More formally, the logarithm of the probability density function of $\mathbf{x}_y | y$ under this generative model is

$$\log(P(\mathbf{x}_y | y, \mathbf{\mu}_y, \Lambda_y, \nu)) = \sum_{j=1}^{d_y} \log(P(\mathbf{x}_{y,i} | y, \mathbf{\mu}_{y,i}, \lambda_j, \nu)),$$

where

$$P(\mathbf{x}_{y,i} | y, \mathbf{\mu}_{y,i}, \lambda_j, \nu) \propto \frac{1}{\sqrt{\lambda_j}} \left(1 + \frac{(\mathbf{x}_{y,i} - \mathbf{\mu}_{y,i})^2}{\nu \lambda_j}\right)^{-(\nu+1)/2} \quad \text{for } j = 1, \ldots, g_y \quad (3-13)$$

Differentiating the left hand side of (3-13) with respect to each of the model parameters, we obtain the score vector as:

$$\varphi(\mathbf{x}) | y, \mathbf{\mu}_y, \Lambda_y = [\varphi_{\mu,1}, \ldots, \varphi_{\mu,g_y}, \varphi_{\Lambda,1}, \ldots, \varphi_{\Lambda,g_y}]', \quad \text{where for } j = 1, \ldots, g_y$$

$$f_{\nu,j}(\mathbf{x}) = \left(1 + \frac{(\mathbf{x}_{y,i} - \mathbf{\mu}_{y,i})^2}{\nu \lambda_j}\right)^{-\nu}$$

and

$$\varphi_{\mu,j}(\mathbf{x}) | y, \mathbf{\mu}_y, \Lambda_y = \frac{(\nu+1)(\mathbf{x}_{y,i} - \mathbf{\mu}_{y,i})^2 f_{\nu,j}}{\nu \lambda_j}.$$
The formula for $I_{G_y}$ thereby is:

$$I_{G_y} | y, \tilde{m}_{G_y}, A_{G_y} = diag(I_{\mu,1}, ..., I_{\mu,g_y}, I_{\lambda,1}, ..., I_{\lambda,g_y})$$

where for $j = 1, ..., g_y$

$$I_{\mu,j} | y, \tilde{m}_{G_y}, A_{G_y} = \frac{\nu + 1}{(\nu + 3) \lambda_j},$$

$$I_{\lambda,j} | y, \tilde{m}_{G_y}, A_{G_y} = \frac{\nu \lambda_j^2}{2(\nu + 3)}$$

To execute (3-14) or (3-15), a proper value for the degree parameter $\nu$ is needed. We explore the use of a hierarchical model as a hint to estimate it, as described below.

It has been well studied that, for a Gaussian distributed random variable, if its variance is of an inverted Gamma distribution with parameter set $(\alpha, \beta)$, then it is marginally of a Student-t distribution with same mean, degree $2\alpha$ and scale $\lambda = \beta / \alpha$.

Under this hierarchical model, we can find a heuristic approach to estimating $\nu$ via the training sample under each class. The details are extracted below.

First, we assume the orthogonal projection has been done (after eigenvector elimination). With the discussions in the last subsection, we know that the covariance matrix of this projected dataset is $A_{G_y}$ (as defined in Eq. 3-7b). Under the hierarchical model discussed above, the inverse of the diagonal elements of this matrix is of a Gamma distribution with parameter $(\alpha, \beta)$. With a moment matching procedure, the estimated $\nu$ under class $y$ is given by:

$$\nu = 2(m_{\lambda}^2 / S_{\lambda}^2 + 2),$$

where
\begin{equation}
    m_\lambda = \frac{1}{g_y} \sum_{j=1}^{g_y} \lambda_j \quad \text{and} \quad S^2 = \frac{1}{g_y - 1} \sum_{j=1}^{g_y} (\lambda_j - m_\lambda)^2
\end{equation}

(3-16)

The $\lambda_j$ for $j = 1, \ldots, g_y$ in (3-16) is defined in (3-7b). Note that, if $g_y$ is very small, (3-16) will not provide a robust estimator for $\nu$ (or this formula even does not work at all). Therefore, we do not recommend using the kernel suggested in this section, if $g_y$ for any class is smaller than 3. In this study, for simulation comparison purpose, we apply $\nu = n_y - g_y$ when $g_y < 3$.

With (3-16), we have built a natural kernel with the multiple-univariate Student-t distribution as the generative model for the gene expression data of the significant gene set. Throughout this study, we shall refer to this kernel as natural Student-t Fisher kernel (N1FK). The detail procedure is listed below:

The initialization procedure is the same as that in algorithm (3-12). For each class, we execute the following steps.

1. - 5. The first 5 steps are same as those in algorithm (3-12).
6. Calculate the Fisher information matrix $I_\nu$ via (3-15).
7. Estimate $\nu$ via (3-16).
8. Row by row for the projected training dataset, calculate its Fisher score via (3-14) to generate a score matrix $\Phi_{\nu}$ for the training set, each column of which contains the score vector of a training sample.
9. For each row of the projected testing dataset, also calculate its Fisher score via (3-14) to generate a score matrix $\tilde{\Phi}_{\nu}$ for the training set, each column of which contains the score vector of a testing sample.
10. - 12. Steps 10 through 12 are the same as steps 9 through 11 in algorithm (3-12).

(3-17)

The discussion of NTFK building algorithm in (3-17) is almost the same as that for the algorithm in (3-12). An important practical issue (as we mentioned before) is that, we do not recommend using NTFK for a real application, if for any class, the number of compressed feature for training set, $g_y$, is smaller than 3.

3.2.3 Issues on Implementation

Although the NK (NGFK and NTFK) building algorithm presented in this section can be seamlessly embedded into the KIGP framework introduced in Chapter 2, we do not recommend apply it to a “gene selection phase” of a KIGP. The reason is as follows. NK building in essence is a data driven learning process because parameters of the presumed generative model are estimated based on the input training data. The performance of an NK-based analysis therefore heavily relies on the correctness of the training data in model. Furthermore, there is a feature compression (orthogonal projection) step in the procedure of building an NK. As the result, an NK imbedded learner is very powerful in self-adapting to the predictor (gene) set in the model. These two characters of building an NK imply that an NK is much less sensitive to non-significant (false positive) genes than a regular kernel type such as LK or GK. This means that an NK-based learning method is more robust provided a set of significant genes. But NK is not appropriate to be part of a gene-selection algorithm with wrapper style such as a KIGP.
This basically is a tradeoff between performance and generality. We gain extra performance by extracting features of the training data to build a kernel, while we also would lose some beneficial functions of the framework. Actually, if we use an NK in a KIGP simulation within a gene selection phase, the MCMC algorithm could be easily trapped into some locally stationary state. It usually needs much more for the MCMC algorithm iterations to converge or the algorithm would never converge at all. We shall illustrate this phenomenon in example 2 of this Chapter.

As we have discussed above, NK basically is a data driven model, building an NK into the gene-selection phase of a KIGP is not appropriate. Below, we describe a univariate ranking (UR) algorithm to execute the gene-selection procedure for a NKIGP. The general assumption adopted here is that, for most real applications, significant genes usually should contain more information with respect to class variation. Thus, It is safe to eliminate those genes that have almost the same gene expression level through samples from different classes. In this study, we apply the UR method proposed in [16]. In details, the ranking score for each gene is defined as the ratio of the weighted sum of within-class variance and the global variance:

\[
\rho_j = \frac{\sum_{m=1}^{M} n_m (\bar{x}_j^{(m)} - \bar{x}_j)^2}{(n - M) \sigma_j^2} \quad \text{for } j = 1, 2, ..., p , \text{ where}
\]

\[
\bar{x}_j^{(m)} = \frac{1}{n_m} \sum_{i, y_{im} = m} x_{ij} , \quad \sigma_j^{(m)^2} = \frac{1}{n_m - 1} \sum_{i, y_{im} = m} (x_{ij} - \bar{x}_j^{(m)})^2
\]

and \[
\bar{x}_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij} , \quad \sigma_j^2 = \frac{1}{n - M} \sum_{i=1}^{n - M} (x_{ij} - \bar{x}_j)^2
\]  
(3-18)

In (3-18), the symbol \( n_m \) represents the number of training samples in class \( m \).
After applying the UR method in (3-18), we should obtain a set of ranking scores for each gene in a model. To set a threshold of the score to pick up the significant gene set, rather than resorting a cross-validation approach, we adopt the empirical Bayes method suggested by [17]. The procedure is similar to that shown in section 2.1.5.2 for the KIGP gene-selection.

We first normalize the score set so that mean and variance of the normalized score is 0 and 1 respectively. Then, we apply the same gene selection procedure as in section 2.1.5.2 to this normalized set to set a threshold for the normalized score under some confidence level. In the end, we pick the genes above the threshold. Compared to the method in section 2.1.5.2, the only difference is that the target statistic here is the normalized UR score. Throughout this study, we use 0.05 as the significance level for UR score. After identifying the significant gene set, we thereby can directly apply the “prediction phase” of a KIGP to this set. Based on the arguments addressed in the start of this section and the simulation results of simulated examples, in this study, we shall adopt the UR method described above as the gene-selection scheme for an NKIGP.

On a side note, rather than KIGP, the NK building method introduced in this section can as well be applied to most kernel-related classifiers without modification. For example, [76] suggests an algorithm for analyzing a multi-classification model using a Gaussian Process classifier. They built a model with all interception \( b_m \) set to 0 and variance \( \sigma^2 \) set to 1 and calculated the Bayesian inference for the posterior distribution \( P(t \mid X, y) \) by using a Laplace approximation (terms used here are defined in Eq. 2-23). The natural kernel can be seamlessly adopted in that algorithm. The similar Laplace approximation method actually is also applicable to the KIGP model in this study (probit
regression based). With $\sigma^2 = 1$, an iterative algorithm with guaranteed convergence is available to find the approximated mean and variance for the feature vectors $\mathbf{t}$ and $\mathbf{b}$. However, due to the non-linear set of class decision function in a multinomial probit regression model (so is KIGP), the resulted iterative calculations are very complicated. Hence, with a KIGP, we still recommend using the MCMC algorithm proposed in Chapter 2 (prediction phase of a KIGP) to estimate the mode of the posterior distribution of model parameters, as well the predictions of testing samples.

### 3.3 Simulation Results and Discussions

In this section, we present a few simulated examples as well several real case studies. In order to compare NK with other regular kernel types (such as those defined in Eq. 2-1) and other classical methods referred in Chapter 2, for the sake of simplicity, we only revisit the examples that have been already used in the last two chapters. In the simulated examples, we focus on the illustration of the correctness and effectiveness by using an NK. In the real data experiments, we present the results by applying NKIGP to the datasets. A fair performance comparison to other state-of-the-art microarray analysis methods is also made. The detail descriptions of all the referred methods and the performance measure procedures are found in section 2.3.
3.3.1 Simulated Examples

Example 1: A binary classification problem

In this example, we demonstrate that an NKIGP can closely achieve an optimum performance without a kernel parameter tuning procedure. This example includes two cases. In the first case, the Bayesian bound can be well approximated by a linear function, whereas in the second case the Bayesian bound takes a very non-linear form. Based on the results of the independent testing procedure (Table 3-1), one can see that KIGP with an NK can automatically adapt to a fitted feature space in both the linear case and the nonlinear case due to its feature-compression embedded kernel building procedure, whereas KIGP with a GK realizes feature space scanning via tuning its width parameter.

<table>
<thead>
<tr>
<th>Kernel Type</th>
<th>LK</th>
<th>Optimum GK</th>
<th>NGFK</th>
<th>NTKF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1 (linear)</td>
<td>0.12</td>
<td>0.08</td>
<td>0.08</td>
<td>0.08</td>
</tr>
<tr>
<td>Case 2 (non-linear)</td>
<td>0.30</td>
<td>0.14</td>
<td>0.15</td>
<td>0.15</td>
</tr>
</tbody>
</table>

Table 3-1: Performance comparison for Example 1. The values are the resulting MR for each classifier.
Fig. 3-1: **Class decision function plot for case 1 of example 1.** The asterisks denote training samples in class “1”, whereas the circles represent the training samples in class “-1”. The solid lines show the optimum (Bayesian) decision function. In each panel of the figure, the red dots and green dots mark class “1” and “-1”, respectively, of the testing samples decided by the KIGP output with different kernels.

**Case 1: A linear case**

The first case is actually a revisit of example 3 in section 1.2.5. The generative model, the training set and the testing set are the same as described in that section. The detailed description of the generative model can be found therein as well. In that example, we applied KIGP with different GK (with different kernel width) to the training set. The independent testing prediction performances for each KIGP are provided in Fig. 1-7. The resulted decision functions (per mean of the KIGP outputs) with 4 different kernel widths
are drawn in Fig. 1-6. KIGP/GK(1.0) delivered the best predictive performance for the testing set.

In this case, we applied KIGP with an NGFK and an NGTK, respectively, to the training set. As a benchmark comparison, we also applied KIGP with an LK to this training set as well. The experimental procedure of this case is the same as that of its counterpart example in Chapter 1. The resulted misclassification rate for each classifier, along with that of the best KIGP performer with a GK (KIGP/GK(1.0)) are listed in Table 3-1. Their associated class decision functions for each KIGP are plotted in Fig. 3-1. Comparing all the kernel types examined in this case, one can see that KIGP with any of these kernels functioned very well for this linear case. The output class decision functions for the KIGPs with each kernel are very similar. Actually they are all very close to the underlying Bayesian classifier.

Case 2: A non-linear case

This case is a revisit of example 2 in section 1.2.5. Again, the generative model, the training set and the testing set of this example are exactly the same as those of example 2 in Chapter 1. In that example, we applied KIGP with different GK to the training set and the testing prediction performances for each KIGP are provided in Fig. 1-5. The resulted class decision functions with 4 different kernel widths are drawn in Fig. 1-4. The best KIGP performer with a GK for this non-linear case is KIGP/GK(0.45).
Fig. 3-2: Class decision function plot for case 2 of example 1. The asterisks denote training samples in class "1", whereas the circles represent the training samples in class "-1". The solid lines show the optimum (Bayesian) decision function. In each panel of the figure, the red dots and green dots mark class "1" and "-1", respectively, of the testing samples decided by the KIGP output with different kernels.

In this case, we applied the same experimental procedure as we did in case 1. The resulted misclassification rate for each classifier, along with that of the best KIGP performer with a GK (KIGP/GK(0.45)) are provided in Table 3-1. The output class decision functions for each KIGP are drawn in Fig. 3-2. Obviously, LK is not appropriate for this non-linear example. With a GK, as shown in Example 2 of Chapter 1, KIGP/GK(0.45) delivered a class decision function that is reasonably close to the Bayesian classifier (Fig. 3-2). But in order to find the best fitted kernel width, a reliable and independent testing set is usually needed. Otherwise a rigorous CV procedure (such
as the 3-fold CV described in section 2.3.1) has to be applied. By contrast to GK, NK (NGFK or NGTK) is non-parametric (like LK). Because NK is built on the extracted feature of the training set, it is not surprising that in this example KIGP with either an NGFK or an NGTK performed very close to KIGP/GK with the best fitted width (Table 3-1). The resulted class decision function of KIGP with an NK is again very close to the Bayesian classifier (Fig. 3-2).

In summary, from case 1 and case 2 of this example, we can see that, without the need of tuning kernel parameter, KIGP with an NK can adaptively discover the underlying fitted feature space in both the linear case and the non-linear case, yielding a performance that is very close to the Bayesian bound. On the one hand, as Occam’s razor states, if there are two models that both can explain a set of training samples well, the simpler one should be more preferable than the complicated one. Therefore, compared with a parametric kernel type such as GK, NK is advantageous because it contains fewer parameters. On the other hand, although LK is also a non-parametric kernel, as we have illustrated in this example, it does not work well for a problem with a very non-linear Bayesian classifier. By contrast, NK delivered very satisfactory results for both the linear case and the non-linear case.

**Example 2: A multi-classification problem**

This example is a revisit of the multi-classification simulated examples in Chapter 2. Specifically, there are 3 cases in this example. The first case (case 1) refers to example 1 in section 2.3.2.2; the second case (case 2) refers to example 2 in section 2.3.2.2; the third case (case 3) refers to the example in section 2.3.2.3. In this example, we discuss
two issues when using an NKIGP. First, similar to example 1 in this section, we illustrate the effectiveness and correctness of a NKIGP classifier for a multi-classification problem by comparing it to the KIGP with a regular kernel as in Chapter 2. Second, by separately applying the two gene-selection approaches (KIGP approach and UR approach) to the training set, we shall discuss the advantages and the disadvantages of using either method and provide evidence of why we shall adopt the UR approach as our gene-selection scheme under NKIGP framework.

Fig. 3-3: Class decision function plots for example 2. The asterisks mark the training samples in class “1”; the triangles demonstrate the training samples in class “2”; the circles show the training samples in class “3”, given the two prescribed significant genes (23, 57). The dotted curves are the resulted class decision functions for each simulation. The solid lines display the underlying Bayesian classifiers. MR represents misclassification rate for the testing set. In figure (e) and (f), the mislabeled sample is overlapped with both an asterisk and a triangle.
The generative models, the training sets and testing sets used in this example are exactly the same as those in the referred examples in Chapter 2. The details of the example settings can be found therein. In a rough description, case 1 is with a semi-linear Bayesian classifier between each pair of classes; case 2 is with a very non-linear Bayesian classifier between class “1” and “2”; case 3 is same as case 1 except that one of the training samples in class “1” is intentionally mislabeled.

![Graphs showing intermediate results of the gene-selection phase for example 2.](image)

Fig. 3-4: **Intermediate results of the gene-selection phase for example 2.** The dotted lines display the threshold for the ranking score and the dots mark the genes that are selected. For KIGP ranking score NLF, the significance level is 0.05; for the normalized UR score, the significance level is 0.1.

Fig. 3-3 shows the output results by applying KIGP with an NK to the training set and the testing set in each case with the assumption that the two underlying significant genes are already known. Comparing to the results in their counterpart examples in...
Chapter 2, we can see that KIGP/NK (either NGFK or NGFK) performed well for all these cases. Actually their performance was very close to KIGP with the optimum GK (Fig. 2-6, Fig. 2-8 and Fig 2-10). Clearly, the associated class decision functions in Fig. 3-3 are also good approximations for the underlying Bayesian classifiers.

The intermediate results for the gene selection phase of each simulation are shown in Fig. 3-4. By running KIGP/NK in “gene selection phase”, we obtained the relative NLFs for each gene, based on which we could select the significant gene set. For case 1 (liner), both NGFK and NGFK still could successfully identify the two true significant genes (gene 23 and 57). However, for case 2 (non-linear) and case 3 (linear, with a mislabeled training sample), KIGP/NK ranking was not successful, especially when compared to the KIGP with a GK.

The UR method could retrieve both significant genes in case 1 and case 3. Even in the non-linear case (case 2), UR still found the two true significant genes, although it also made one false positive in the gene selection. Obviously, comparing to the KIGP/NK gene selection procedure, the UR method is much more robust in identifying significant genes in a problem, especially in those cases that either the Bayesian classifier cannot be well approximated by linear functions or the provided training samples are not consistent.

One finding of this example deserves being highlighted here. That is, applying KIGP/NK with UR as the gene-selection strategy has an outstanding advantage: it is much less sensitive to false positive genes. For instance, in case 2 of this example, per the UR method, we selected 3 significant genes, one of which (gene 160) is a false positive for the true generative model. However, even with this “mistaken” gene, the KIGP with an NK still performed reasonably well. In both the NGFK simulation and the NTFK
simulation, the resulted testing MR was 0.15. With these three genes, the testing MR of the KIGP/GK simulation is 0.20. We also can compare KIGP/NK to all the results of the simulations with a GK for this case in Chapter 2 (Table 2-1). It is quite obvious that NK is better (or, say, more robust to false positive) than all of other methods therein (SVM/GK, PLR/GK).

In summary, this example gives us a hint on the fact that, rather than a wrapper method, it is better to use a simpler filter approach, such as the UR method in (3-18) for gene selection when using an NK because NK in nature is data driven. A Monte Carlo procedure could easily lead a KIGP/NK overfit some wrongly selected genes. Using a UR approach is obviously more robust. This example as well demonstrates the procedure for applying a KIGP/NK: we first use the UR method in (3-18) for gene selection; afterward we apply KIGP (prediction phase only) to the selected gene set.

### 3.3.2 Real Data Studies

In this section, we apply the KIGP/NK framework to the published datasets studied in section 2. The descriptions for all these datasets are detailed in subsection 2.3.3. The overall results are provided in Table 3-2.

According to the UR gene selection procedure described in this chapter, we used the top 10 genes for the analysis for the leukemia dataset; we used the top 10 genes for the SRBCT dataset; we used the top 29 genes for the colon dataset; we used the top 40 genes for the lymphoma dataset; we used the top 24 genes for the breast cancer dataset; and we used the top 63 genes for the brain dataset. One can see that for most datasets, both NGFK and NTFK performed fairly well.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Kernel Type</th>
<th>LOOCV # of Err</th>
<th>Test # of Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leukemia</td>
<td>NGFK</td>
<td>0/38</td>
<td>4/34</td>
</tr>
<tr>
<td></td>
<td>NTFK</td>
<td>0/38</td>
<td>4/34</td>
</tr>
<tr>
<td>SRBCT</td>
<td>NGFK</td>
<td>0/35</td>
<td>0/12</td>
</tr>
<tr>
<td></td>
<td>NTFK</td>
<td>0/35</td>
<td>1/12</td>
</tr>
<tr>
<td>Colon</td>
<td>NGFK</td>
<td>5/62</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>NTFK</td>
<td>5/62</td>
<td>NA</td>
</tr>
<tr>
<td>Lymphoma</td>
<td>NGFK</td>
<td>0/62</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>NTFK</td>
<td>0/62</td>
<td>NA</td>
</tr>
<tr>
<td>Breast Cancer</td>
<td>NGFK</td>
<td>0/22</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>NTFK</td>
<td>0/22</td>
<td>NA</td>
</tr>
<tr>
<td>Brain Tumor</td>
<td>NGFK</td>
<td>1/42</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>NTFK</td>
<td>1/42</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3-2: Testing results for the real datasets by using NKIGP. In the Test column, the format is “# of errors / # of testing samples” and “NA” means “not available”.

In the case for the brain tumor dataset, comparing to the relative results in Chapter 2, KIGP/NK made a mistake in LOOCV prediction for the sample #40 with a small margin. Another interesting finding is that both KIGP/NKs made 4 testing errors for the leukemia dataset. This is actually very similar to what was reported by the original authors ([23]), who also adopted a UR method for the gene selection. Our results here are actually very consistent to theirs. In the case for the SRBCT dataset, the APP of the 12 testing samples with a KIGP/NGFK is 0.91, which is almost the same as that for the best
KIGP with a regular kernel (KIGP/PK(1) with an APP of 0.92). Overall, the only dataset for which KIGP/NK made significant number (5 out of 62) of “errors” is the colon dataset. We therefore focus on this “non-trivial” dataset again and explore it more in depth.

<table>
<thead>
<tr>
<th>Method</th>
<th>KIGP/NGFK</th>
<th>KIGP/NTFK</th>
<th>KIGP/GK</th>
<th>BagBoost</th>
<th>Boosting</th>
<th>RF</th>
<th>SVM</th>
<th>GPARD</th>
<th>PAM</th>
<th>DLDA</th>
<th>kNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR</td>
<td>0.108</td>
<td>0.117</td>
<td>0.129</td>
<td>0.161</td>
<td>0.191</td>
<td>0.159</td>
<td>0.151</td>
<td>0.201</td>
<td>0.119</td>
<td>0.129</td>
<td>0.164</td>
</tr>
</tbody>
</table>


As seen in Chapter 2, based on a multiple rigorous CV procedure, a KIGP with a GK was among the best performers. However, it also made 5 prediction errors in the LOOCV test and the index of the 5 mislabeled samples is #18, #20, #45, #49 and #56 (refer to Fig. A-3). We therefore suspect these 5 samples are mistakenly phenotyped by the original authors. In this section, obviously not a coincidence, we found that these 5 samples were exactly the 5 mislabeled ones in the LOOCV test with very significant margin of errors. This reinforces our belief that these 5 samples are mislabeled originally. Based on this fact, now we can understand why most linear methods do not perform well for the colon dataset. Below we shall first revisit the case for this dataset in Chapter 2 and then add our new findings into the simulation results therein. The detailed procedure for the other methods is found in section 2.3.3.3. In this section, in order to make a fair comparison to the other classifiers, we also chose the top 200 genes based on Wilcoxon
statistic to do the simulations with KIGP/NK. The overall performance comparison of different classifiers for this dataset is organized in Table 3-3. Quite obvious, KIGP/NGFK is by far the best method for this dataset that contains multiple suspiciously mislabeled training samples. It is also significantly better than KIGP with a GK.

In Fig 3-5, we demonstrate that the KIGP/NK approach comparatively is not very sensitive to the number of genes selected in the model based on the UR score. We applied the KIGP/NK method with different number of top significant genes to the dataset with a multiple (50 in this example) rigorous 3-fold CV procedure detailed in subsection 2.3.1. We can see that KIGP/NK (both NGFK and NTFK) achieved their best performance for a gene number that is between 10 and 30. As a benchmark comparison, we also applied the best (adjusted) linear method, DLDA, to this dataset with the same experimental settings. According to Table 2-9, DLDA is one of the best performers for this colon dataset. However, in Fig. 3-5, we can see that KIGP/NK delivered noticeably better performance. In this example, we can see that KIGP/NK is a very robust and efficient learner for a dataset that a linear learning method works flakily.

In summary, generally speaking, the NKIGP framework presented in this chapter is essentially based on a feature orthogonal projection procedure with respect to the most informative genes under investigation. In this section, by adopting a UR method as the gene selection strategy, we have applied the KIGP/NK framework to six published datasets that have been tested by KIGP with a regular kernel in Chapter 2. For the colon dataset, the SRBCT dataset, the breast cancer dataset and the lymphoma dataset, very satisfactory results have been achieved. The reason why KIGP/NK did not perform very well for the leukemia dataset (same as in [24]) is possibly because some critically
significant genes for the target problem do not have a high UR score, which implies that some of the true significant gene clusters may have very high correlations between each other. For the only non-trivial dataset of this study, the colon dataset, KIGP/NK shows its outstanding performance, demonstrating its promising potential for analyzing a dataset containing inconsistent information.

Fig. 3-5: MR plot by applying NKIGP with different number of significant genes to the colon dataset. DLDA refers to "Diagonal Linear Discriminant Analysis". The bound line is based on the assumption that there are 5 mislabeled samples for this dataset.
Chapter 4

Simultaneous Kernel Type Competition:

An RJMCMC Approach

As we have thoroughly discussed in this study, the kernel function uniquely decides the property of a KIGP's induced feature space, thus its generalized performance. Via a variable selection approach and assuming a regular kernel type has been given, we develop a complete Bayesian framework, KIGP, for microarray analysis in Chapter 2. Under a natural kernel frame, we built another kernel-induced microarray learning approach, NKIGP, by adopting an orthogonal feature projection in the kernel building procedure. In the simulation result parts of these two chapters, we have demonstrated that both methods are very effective to capture the characteristic feature of a target microarray analysis problem.

Nonetheless, for the KIGP framework developed in Chapter 2, we still face the question, "how to appropriately select the best kernel type from a list of candidates?". As seen in Chapter 3, deciding whether we should adopt an NGFK or an NTKFK is also a very practical problem. Furthermore, even though we already have had a winner kernel type under either the KIGP framework (in Chapter 2) or the NKIGP method (in Chapter 3), we
also need to decide which one would deliver better generalized performance for the target problem. Obviously, all these questions can be summarized as a kernel type competition problem.

For a general model competition problem, some simple approximate criteria, such as AIC, BIC, are available. However, it turns out they are not effective at all for a kernel type comparison problem because their simple underlying assumption cannot catch the deliberate difference of different kernels. For this reason, we present the details of a predictive fit measure approach in subsection 2.3.1 to deal with the kernel type competition problem. Although this simple approach is very reliable when a sufficient number of independent testing samples are available, it nonetheless has to resort to a rigorous cross-validation (CV) procedure to fulfill a credible model generalized performance assessment when the number of samples of the target application is limited. The rigorous CV suggested in subsection 2.3.1 is especially computationally expensive for our proposed KIGP framework, because the core of it essentially is an MCMC algorithm.

In most real microarray applications, the number of available samples is almost always limited, usually ranging from 10 to a few tens. Hence, a rigorous cross-validation procedure is usually the only option. There is another problem to apply the predictive measure approach to the kernel type competition problem under a KIGP framework. That is, first for the KIGP method in Chapter 2, the number of the investigated kernel types could be large; second, for the NKIGP algorithm in chapter 3, we also could extend the candidate generative distribution model from "Gaussian or Student-t" to a much wider range of distribution models. In order to implement a predictive measure approach, one
has to run multiple simulations under each of the considered kernel type. As a result, it is almost computationally prohibitive to adopt the predictive measure method for the kernel type competition problem. Under the consideration of these two obstacles for implementing a predictive fit measure approach to kernel type selection, we were motivated to find a more efficient method for the kernel type competition problem under a Bayesian framework. A simultaneous kernel type competition algorithm is definitely a crucial issue for the practical implementation of the method proposed in this study.

Under the overall KIGP framework introduced in Chapter 2 and Chapter 3, different kernel types do not share the same kernel parameter set (if applicable) and their associated model parameters. The dimension of parameter space for different kernel type usually is not the same either. Therefore, a regular MCMC algorithm is not able to function for the kernel type competition problem. This dimension-mismatching problem almost always exists in model selection/competition problems. In order to deal with this issue a few approaches have been suggested in the past years. One of the most useful established methods in this context was the reversible jump MCMC (RJMCMC) algorithm, which was first published in [25]. Later, an enhanced version of the RJMCMC algorithm is introduced in [22]. In this study, we shall adopt the general RJMCMC algorithm to our proposed KIGP framework. With some proper approximation, a RJMCMC algorithm is specially designed for the kernel type competition problem of this study, based on which an algorithm will be provided in the end of this chapter to unify the methods introduced in Chapter 2 and Chapter 3 under a framework.

The rest of this chapter is organized as follows. In section 4.1, we introduce the mathematical background for the general RJMCMC algorithm. In section 4.2, we present
the specially designed RJMCMC algorithm for simultaneous kernel type competition problem under the KIGP/NKIGP framework proposed in this study. An overall algorithm as a conclusion of this study is provided in section 4.3, along with a few simulated examples and real data case studies to testify the correctness and effectiveness of the proposed method.

4.1 Introduction to the General RJMCMC Method

The general RJMCMC method is a generalized extension of the regular MCMC algorithm. A brief overview of the MCMC method is given in section 1.2.3. The symbols and terms used in this section are the same as those therein, except that in this section for the kernel type competition problem, each of the likelihood distributions is conditional on a specific kernel type. Throughout this chapter, we term a kernel type by the symbol “$H$”.

As before, we assume that $\theta$ is the set of the model parameters under kernel type $H$. The fundamental Bayesian model is thus ruled by the following formula:

$$P(\theta \mid X, H) = \frac{P(X \mid \theta, H)P(\theta \mid H)}{\int P(X \mid \theta, H)P(\theta \mid H)d\theta} = \frac{P(X \mid \theta, H)P(\theta \mid H)}{P(X \mid H)}$$

Here, the symbol $X$ denotes the training dataset. For a supervised learning problem, it implicitly includes the response label set as well. Note that the term on the denominator of this formula, the marginal distribution of data $P(X \mid H)$ (it is also known as the evidence of the model $H$) is not dependent on the parameter set $\theta$. Therefore the evidence is not of our interest, if we only need to estimate probabilities of the parameters with a given model $H$, such as the cases in Chapter 2 and Chapter 3. However for the...
kernel type competition problem, this quantity is critical since the posterior probability of the model, \( P(H \mid X) \), is proportional to the evidence, \( P(X \mid H) \), multiplied by the prior for the model, \( P(H) \). The RJMCMC algorithm was developed to retrieve the posterior probabilities of all investigated model through a Monte Carlo simulation de facto.

Generally speaking, the RJMCMC algorithm is a Metropolis-Hastings type algorithm ([18]) that moves a simulation analysis between different models based on an accept-reject scheme. Suppose that candidate models are enumerable and represented by the set \( H = \{ H_0, H_1, ..., H_{K-1} \} \). We denote the joint set of the relative parameters within model \( H_k \) by \( \theta_k \), for \( k = 0, 1, ..., K-1 \). The key of a RJMCMC algorithm is that one further introduces a random vector \( u_k \), \( k = 0, 1, ..., K-1 \), such that for any \( k' = 0, 1, ..., K-1, k' \neq k \), the dimension of \( \{ \theta_k, u_k \} \) and \( \{ \theta_k', u_k' \} \) can be matched. Then one sets \( \theta_k \) to be a deterministic function of \( \theta_k \) and \( u_k \). For the reversible jump move, one proposes a vector \( u_k \) and then set \( \theta_k \) to be a deterministic function of \( \theta_k \) and \( u_k \). Thus, there must be a bijection between the set \( \{ \theta_k, u_k \} \) and the set \( \{ \theta_k', u_k' \} \). This mapping function is defined by \( (\theta_k', u_k') = g_{k,k'}(\theta_k, u_k) \). A general RJMCMC algorithm within each iteration (the observation dataset \( X \) is assumed to be given by default) can be sketched as below (after initialization and assuming the currently accepted model is \( H_k \)):

1. Propose a visit to model \( H_{k'} \), \( k' \neq k \), with probability \( J(H_k \rightarrow H_{k'}) \).
2. Sample \( u_k \) from a proposal density \( Q(u_k \mid \theta_k, H_k, H_{k'}) \).
3. Set \( (\theta_k', u_k') = g_{k,k'}(\theta_k, u_k) \).
4. Calculate the odds ratio $r$ and accept $H_k$ as the model with probability 

"$\text{min}(1, r)$", where

$$r = \frac{P(X | \theta_k, H_k)P(\theta_k | H_k)J(H_k \to H_k)Q(u_k | \theta_k, H_k, H_k) \partial g_k \cdot k}{P(X | \theta_k, H_k)P(\theta_k | H_k)P(H_k)J(H_k \to H_k)Q(u_k | \theta_k, H_k, H_k) \partial (\theta_k, u_k)}$$

If $H_k$ is rejected, we maintain the current model $H_k$.

5. Return to the step 1.

(4-1)

After a proper burn-in period, the number of times that model $H_k$ remains in the simulation (after step 4 in (4-1)) divided by the total number of iterations will be a consistent estimation to the posterior probability of $H_k$, $P(H_k | X)$. Furthermore, the samples drawn from each iteration with the model $H_k$ are equivalently drawn from the posterior joint distribution $P(\theta_k, H_k, H_k)$, which can be used for further analysis within model $H_k$.

In the general RJMCMC algorithm (4-1), designing an appropriate matching random vector $u_k$ and the associated proposal probability function $Q(u_k | \theta_k, H_k, H_k)$ is crucial for the efficient implementation of this algorithm. It is usually very constructive and flexible. The key for building an efficient proposal probability function $Q(u_k | \theta_k, H_k, H_k)$ is to make it close to the posterior distribution $P(u_k, \theta_k | X, H_k)$. In the following section, we shall design an algorithm for the KIGP framework following this guideline.
4.2 RJMCMC Designed for the Kernel Type Competition

Problem

We have introduced the general RJMCMC algorithm in the last section. In this section, we shall develop an approach to kernel type competition for the microarray analysis method proposed in this study, with a RJMCMC algorithm as the core. The method is particularly designed under two different scenarios. Under the first scenario, we assume that we have already fixed the kernel parameters and the gene selection vector under all investigated kernel types. That is, under the KIGP framework in Chapter 2, the first two phases are over and the only phase left is the "prediction phase"; with the NKIGP method suggested in Chapter 3, the UR gene selection procedure has been done and a set of significant genes has been chosen. Under the second scenario, we refer to the first two phases (kernel parameter fitting phase, if applicable, and gene-selection phase) in the KIGP framework in Chapter 2. That is, the kernel parameter and the gene selection vector are still undetermined. We refer to the first scenario as "RJMCMC for prediction phase" and term the resulting algorithm by "KIGP_RJ_1"; and we refer to the second scenario as "RJMCMC for pre-prediction phase" and term the resulting algorithm by "KIGP_RJ_2". We shall merge the NKIGP model in Chapter 3 with the KIGP framework in Chapter 2 with the algorithm "KIGP_RJ". The summary general algorithm for this dissertation will be outlined in the end of this chapter.

Before presenting our simultaneous kernel type competition algorithm, we shall first introduce a specific version of the RJMCMC algorithm. Referring to the odds ratio formula in the general algorithm (4-1), if the posterior distribution \( P(\theta_k \mid X, H_k) \) under each model \( H_k \) has an analytical close form or can be well approximated, we can set the
proposal density $Q(u_k | \theta_k, H_k, H_k')$ by $P(\theta_k | X, H_k)$. The resulted odds ratio $r$ thus is reduced to

$$r = \frac{P(X | \theta_k, H_k) P(\theta_k | H_k) P(H_k | H_k') J(H_k \rightarrow H_k') P(\theta_k | X, H_k)}{P(X | \theta_k, H_k) P(\theta_k | H_k) P(H_k | H_k') J(H_k \rightarrow H_k') P(\theta_k | X, H_k')}$$  \hspace{1cm} (4-2)

Furthermore, we shall assume a non-informative prior, a uniform distribution, for the choice of kernel type. That is, $P(H_k)$ does not depend on $k$ (uniform). Since there is no overlap parameter space for different kernel types, we do not have a preferred “next-step” move under a given accepted model. Therefore, a uniform distribution is a proper set for the transition probability $J(H_k \rightarrow H_k')$ for any pair of $k$ and $k'$. That is, if the current accepted model is $H_k$, the probability of the kernel type that will be visited in the next iteration is the same for all the candidate kernel types (except $H_k$). With these two settings, the odd ratio in formula (4-2) can be further simplified as:

$$r = \frac{P(X | \theta_k, H_k) P(\theta_k | H_k) P(\theta_k | X, H_k)}{P(X | \theta_k, H_k) P(\theta_k | H_k) P(\theta_k | X, H_k)}$$  \hspace{1cm} (4-3)

In this study, we shall adopt this form of the RJMCMC algorithm. The key would be to properly approximate the posterior distribution for parameter set of a KIGP with each candidate kernel type, $P(\theta_k | X, H_k)$. We shall separately introduce the algorithm under the two scenarios in the following sections, starting with the simpler case, “RJMCMC for prediction phase”.

4.2.1 Scenario 1: RJMCMC for Prediction Phase

There is no kernel parameter and gene selection vector under this scenario and we refer the resulting sub-algorithm under this scenario is the KIGP_RJ_1 algorithm.
Referring to Fig. 4-1, under a KIGP framework given in Chapter 2, the KIGP_RJ_1 algorithm is only applicable for the last phase, the "prediction phase", and it is after the KIGP_RJ_2 algorithm. Under an NKIGP model detailed in chapter 3, the KIGP_RJ_1 algorithm is after the UR gene selection procedure. Specifically, we assume that before starting the algorithm KIGP_RJ_1, under KIGP with regular kernel a winning regular kernel type has been yielded from the KIGP_RJ_2 algorithm, along with its fitted kernel parameter (if the winner is a parametric kernel type) and an identified set of significant genes. The details of the KIGP_RJ_2 algorithm will be provided in the next section. And also, we assume that before starting the KIGP_RJ_1 algorithm a parallel UR gene selection procedure has been done for the NKIGP algorithm and a significant gene set is ready for building natural kernel.

Based on the description above, before we start a KIGP_RJ_1 algorithm, our candidate kernel types include NGFK, NTFK (after UR gene-selection) and a winning regular kernel type with a fixed kernel parameter set (for parametric kernel type only) and gene selection vector. Therefore, referring to the multi-classification KIGP model in (2-23), the parameter set for any kernel type would only contain the latent vector $\mathbf{z}$, the intercept $\mathbf{b}$ and the universal noise variance $\sigma^2$. In this study, our target variable is always the response label set $\mathbf{y}$. Hence, substituting the specific terms in the Bayesian model of this study into the general odds ratio form for a RJMCMC algorithm in (4-3), we have (since significant microarray data impacts the likelihood and posterior distribution in a KIGP only through its kernel matrix, we replace symbol for the default given significant data $\mathbf{X}_r$ by its associated kernel matrix $\mathbf{K}_r$ in the following formula):

136
Clearly, in order to implement algorithm (4-1) with the odds ratio term defined in Eq. (4-4), we need to evaluate the likelihood \( P(y | \theta_k, K_y, H_k) \), the prior \( P(\theta_k | K_y, H_k) \) and the posterior \( P(\theta_k | K_y, H_k, y) \) for all \( k \). In practice, we actually need to instead calculate the logarithm of \( r \). Thus in the following context, we shall provide the procedure to evaluate the logarithm of the likelihood, of the prior and of the posterior term in (4-4) for any target kernel type.

Assessing the likelihood term is straightforward. Based on the KIGP model in (2-23), the label set \( y \) is uniquely determined by the latent vector \( z \). In the KIGP algorithm (including NKIGP) presented in this study, \( z \) is always drawn based on the training label via the procedure in (2-26). As the result, all posterior samples of \( z \) from a KIGP simulation are always consistent to their associated training label \( y \). In other word, \( P(y | z) \) is always equal to 1 in a KIGP with any kernel type.

The log-prior is \( \log(P(\theta | K_y, H)) = \log(P(z | b, \sigma^2)) + \log(P(b)) + \log(P(\sigma^2)) \). Since the prior for intercept, \( P(b) \), is a constant and is the same for any kernel type, this term can be neglected. The term \( \log(P(z | b, \sigma^2)) \) can be evaluated via Lemma 2.1. Since the dimension of vector \( z \) with all kernel types is the same, we can neglect the constant
part of \( \log(P(z \mid b, \sigma^2)) \). As for \( P(\sigma^2) \), according to (2-4), it is of an inverted Gamma distribution. We assume \( P(\sigma^2) \sim IG(\alpha, \beta) \) for all kernel types, so we also can neglect the constant term in this distribution. Combining all these discussions, we have:

\[
\log(P(0 \mid K, H)) = \log(P(z \mid b, \sigma^2 \mid K, H)) = (\alpha - 1) \log(\sigma^2) - \beta \sigma^2
\]

\[
-\frac{1}{2} \sum_{m=1}^{M-1} \left[ \log(\det(K_{m} + \Omega)) + (z_m - b_m \mathbf{1}_n)'(K_{m} + \Omega)^{-1}(z_m - b_m \mathbf{1}_n) \right]
\]

(4-5)

In (4-5), \( K_{m} \), \( \Omega \), \( \mathbf{1}_n \) are same as defined in (2-24); and throughout this study, we always set \( \alpha = \beta = 0.1 \).

To complete the calculation in (4-4), the key is to find a close formula for the posterior \( P(\theta \mid K, H, y) \). Under the KIGP model proposed in this study, it is extremely difficult to obtain an analytical form. A numeric alternative must be used for this context. We hereby adopt an empirical approximation approach and the details are provided as follows.

We first define an adjusted parameter set \( \theta^* \), which is determined by a deterministic function of the original parameter set \( \theta \) and the label set \( y \). More formally, it is defined as:

\[
\theta^* = [z^*_m, b^*_m, \sigma^* \mid m = 1, \ldots, M - 1, i = 1, \ldots, n], \text{ where } \sigma^* = \log(\sigma^2) \text{ and }
\]

\[
b^*_m = b_m, \quad z^*_m = \begin{cases} 
\log(-z_{im}) & \text{if } y_i = M \\
z_{im} & \text{if } y_i = m, \text{ for } i = 1, \ldots, n, \ m = 1, \ldots, M - 1 \\
\log(z_{im}) & \text{otherwise}
\end{cases}
\]

(4-6)

Based on the core algorithm of KIGP in (2-30), each element of the set \( \theta^* \) should always be a real random variable. Based on all the simulations in our study, the empirical
posterior distribution (marginal) for any element of $\theta^*$ is fairly symmetric and can be well approximated by a Gaussian distribution.

After defining $\theta^*$, we introduce a pilot period for the RJMCMC algorithm. That is, we first specifically run the KIGP simulation under each candidate kernel type. After a proper burn-in period, we can draw samples from each simulation. By Markov theory, this is same as directly drawing samples from their joint posterior distribution under each kernel type. Before we start a "true" RJMCMC simulation, we would use the samples drawn from this pilot period to give an initial estimation for the posterior distribution of the adjusted parameter set $\theta^*$. Our assumption is that a multivariate Gaussian distribution can well approximate the posterior distribution of $\theta^*$.

After the pilot period of a simulation, we obtain a set of samples for vector $\theta$, each of which is drawn within each iteration of the pilot period. Then, accordingly, we calculate the set $\theta^*$ from $\theta$ via (4-6). Afterward, we estimate mean and covariance matrix for the random vector $\theta^*$ by using a moment estimation approach. That is, assuming there are $L$ iterations in pilot period and $\theta^{*l}$ is the adjusted samples drawn from the $l$-th iteration, the unbiased estimation for the mean and the covariance matrix of the adjusted set ($m_{\theta^*}$ and $\Sigma_{\theta^*}$) are formulated by

$$m_{\theta^*} = \frac{1}{L} \sum_{l=1}^{L} \theta^{*l}$$
$$\Sigma_{\theta^*} = \frac{1}{L-1} \sum_{l=1}^{L} (\theta^{*l} - m_{\theta^*})(\theta^{*l} - m_{\theta^*})'$$

(4-7)

In a MCMC simulation, we actually do not calculate these two terms through such a batch mode as in (4-7). Instead, we update them on-line within each iteration. That is, we initialize these two terms before the pilot period. As the pilot period begins, we update
them via the deductive formula below, as soon as we obtain a new sample from the simulation.

\[ \mathbf{m}_{\theta}^{[1]} = \theta^{[1]}, \quad \Sigma_{\theta}^{[1]} = \mathbf{0}. \]  
For \( l = 1, 2, \ldots \), \( \mathbf{m}_{\theta}^{[l+1]} = \frac{l \mathbf{m}_{\theta}^{[l]} + \theta^{[l+1]}}{l+1} \) and

\[ \Sigma_{\theta}^{[l+1]} = \frac{l-1}{l} \Sigma_{\theta}^{[l]} + (\mathbf{m}_{\theta}^{[l]} - \mathbf{m}_{\theta}^{[l+1]}) (\mathbf{m}_{\theta}^{[l]} - \mathbf{m}_{\theta}^{[l+1]})^T + \frac{1}{l} (\theta^{[l+1]} - \mathbf{m}_{\theta}^{[l+1]}) (\theta^{[l+1]} - \mathbf{m}_{\theta}^{[l+1]})^T \]

(4-8)

The on-line output from (4-8) is same as that from (4-7) for any \( L \). This on-line updating procedure actually not only is applicable within a pilot period, it also can be implemented in a main period of a RJMCMC simulation.

After the pilot period, we can approximate \( P(\theta^* | \mathbf{K}_r, H, y) \) by a multivariate Gaussian distribution with mean \( \mathbf{m}_{\theta}^* \) and covariance matrix \( \Sigma_{\theta}^* \). In order to obtain the approximation formula for \( P(\theta | \mathbf{K}_r, H, y) \), we need to further consider the determinant of the Jacobi matrix of the transition function in (4-6). Since parameter dimension is the same for all kernel types, we can again skip the constant term. It yields:

\[
\log(P(\theta^* | \mathbf{K}_r, H, y)) \equiv -\frac{1}{2} \log(\det(\Sigma_{\theta}^*)) - \frac{1}{2} (\theta^* - \mathbf{m}_{\theta}^*)^T \Sigma_{\theta}^{-1} (\theta^* - \mathbf{m}_{\theta}^*) - \mathbf{\sigma}_{L}^* - \sum_{\ell=1}^{L-1} \sum_{m=1}^{M-1} \mathbf{z}_{\ell m}^* - \sum_{\ell=1}^{L-1} \sum_{m=1}^{M-1} \mathbf{z}_{\ell m}^*^T \]

(4-9)

In (4-9), \( \theta^* \) is obtained from (4-6); and we always use the most current estimation for the term \( \mathbf{m}_{\theta}^* \) and \( \Sigma_{\theta}^* \) yielded from (4-8) to calculate the log-posterior term in (4-9). If the length of the pilot period \( L \) is set larger than \((M - 1)(n + 1)\), the estimated covariance matrix \( \Sigma_{\theta}^* \) is with full rank. Through (4-5), (4-6), (4-8) and (4-9), after the pilot period,
we can thereby calculate the odds ratio term in (4-4). This makes the KIGP_RJ_1 algorithm usable in practice.

Before listing the pseudo code of the KIGP_RJ_1 algorithm, we first classify the entry input to the algorithm. The candidate kernel type list includes NGFK, NTFK and a winner regular kernel, to which we refer as RK. We actually can extend this list to any other available feature-driven kernel type. RK could be a non-parametric kernel type, such as LK; or it also could be a parametric kernel type, such as GK. We assume for all kernel types, the gene-selection vector has been fixed. For a parametric RK, we assume the associated kernel parameter(s) has been fixed. As the result, the kernel matrix for the training set and testing set has been determined under each candidate kernel type and we refer to it as $K_y$. Overall, assuming the candidate kernel type list is represented by an enumerable set \{H_1, H_2, \ldots\}, the pseudo code of the KIGP_RJ_1 algorithm is listed as follows:

The algorithm consists of 3 successive periods. In order, they are burn-in period, pilot period and main period. The algorithm has two phases, “before main period” (including burn-in period and pilot period) and “main period”.

**Phase of “before main period”:** During this phase, within each iteration, the algorithm is executed as follows:

1. For each of the candidate kernel types, run within prediction of the KIGP algorithm in (2-30).

2. If the simulation in the pilot period, we further execute the following steps:
   a. For each kernel type, sample set $\theta$, including $z$, $b$ and $\sigma^2$.
   b. Convert $\theta$ to $\theta^*$ via (4-6).
c. Update $\mathbf{m}_e$ and $\Sigma_e$ via (4-8).

3. Return to step 1 until the required number of iterations is met.

\[ (4-10a) \]

**Phase of "main period":** After the pilot period, the algorithm enters into the main period. In this phase, first randomly choose a kernel type as the currently accepted kernel type, say $H_k$. With the sample for $\theta_k$ drawn from the last iteration of the pilot period, Set $LP_k = \log(P(\theta_k | \mathbf{K}_\gamma, H_k)) - \log(P(\theta_k | \mathbf{K}_\gamma, H_k, y))$, where $\log(P(\theta_k | \mathbf{K}_\gamma, H_k))$ and $\log(P(\theta_k | \mathbf{K}_\gamma, H_k, y))$ are calculated via (4-5) and (4-9), respectively. After this initialization step, within each iteration in the main period, the algorithm is executed as below:

1. Randomly (with equal probability) choose another kernel type $H_{k'}$, $k' \neq k$.
2. Execute the relevant KIGP algorithm in (2-30) (prediction phase) under kernel type $H_{k'}$, yielding a sample of parameter set $\theta_{k'}$.
3. Convert $\theta_{k'}$ to $\theta_{k'}^*$ via (4-6).
4. Update $\mathbf{m}_{e'}$ and $\Sigma_{e'}$ via (4-8).
5. Calculate the log-prior $\log(P(\theta_{k'} | \mathbf{K}_\gamma, H_{k'}))$ via (4-5).
6. Calculate the log-posterior $\log(P(\theta_{k'} | \mathbf{K}_\gamma, H_{k'}, y))$ via (4-9) (using the updated $\mathbf{m}_{e'}$ and $\Sigma_{e'}$ in step 4).
7. Set $LP_{k'} = \log(P(\theta_{k'} | \mathbf{K}_\gamma, H_{k'})) - \log(P(\theta_{k'} | \mathbf{K}_\gamma, H_{k'}, y))$.
8. Calculate the odds ratio by $r = \exp(LP_k - LP_{k'})$ and accept the kernel type $H_{k'}$ with probability $\min(1, r)$. 

142
9. Return to step 1 until the required number of iterations is met.

10. After the RJMCMC simulation through step 1 to 9 stops, count the frequency of acceptance for each kernel type and pick up the kernel type that appears the most often as the winner kernel type.

\[(4-10b)\]

With a proper set for the lengths of the three periods and after running the algorithm in (4-10), the number of times of a kernel type \(H_k\) accepted by the algorithm divided by the total number of iteration of the main period provides a consistent estimation of the posterior probability of this kernel type, \(P(H_k | X_r, y)\). The winner kernel type picked in step 10 of (4-10b) along with its associated parameter samples of \(\theta\) drawn from the pilot period and the main period can be used for further KIGP analysis and prediction.

In (4-10a), we generally do not need a very long pilot period. A few hundred iterations should be fine for most real applications. Therefore, the overall computation complexity for the phase of “before main period” in (4-10a) is minor. In the phase of “main period” in (4-10b), most computation happens in step 2, which is the same as a regular KIGP algorithm in (2-30). The only extra computation comes from the calculation of the inverse of the matrix \(\Sigma_{\theta_i}\), whose complexity is \(O(n^3)\). However, compared to applying the rigorous CV procedure in section 2.3.1 under each kernel type, it still dramatically improves the efficiency for a KIGP with the kernel type competition problem.
4.2.2 Scenario 2: RJMCMC for Pre-Prediction Phase

The KIGP_RJ_2 algorithm is an extension of the KIGP_RJ_1 algorithm described in the last section. Since an NKIGP only involves the prediction phase of a KIGP, the candidate list for a KIGP_RJ_2 algorithm in this study only contains a set of regular kernel types, which could be non-parametric, such as LK, or parametric, such as GK. Based on the regular KIGP algorithm in Chapter 2, the parameter set for a KIGP_RJ_2 algorithm contains latent vector $z$, intercept $b$, and gene-selection vector $\gamma$ (noise variance $\sigma^2$ is fixed in a KIGP algorithm before "prediction phase"). For a parametric kernel type such as GK, it should include a kernel parameter set as well. In order to accommodate the formulas for the KIGP_RJ_1 algorithm, we use a different symbol $\Theta$ to denote this parameter.

Since different parametric kernel types have different kernel parameters, it is impossible to formulate a universally executable procedure under all regular kernel types. Hence in this study, we alternatively derive the algorithm with only two prototype kernels and then extend the algorithm to other relevant kernel types in the discussion following the algorithm description. Specifically, we develop the KIGP_RJ_2 algorithm for an LK and a GK in parallel. The procedures and formulas for an LK can be applied to any other non-parametric kernel type. The framework for a GK can be used for any other parametric kernel type that only has a scale parameter. In fact, following the principles of building the KIGP_RJ_2 algorithm for a GK, one can effortlessly extend the algorithm to cover any regular parametric kernel. We shall discuss this issue in the end of this section.

Similarly to the KIGP_RJ_1 algorithm, the key step for the KIGP_RJ_2 algorithm is also the proper evaluation of the odds ratio defined in (4-4). With an LK, the parameter
set $\Theta$ contains latent vector $z$, intercept $b$ and gene-selection vector $\gamma$; for a GK it has an extra width parameter for each classifier, $r = [r_1, \ldots, r_{M-1}]'$. That is:

$$\Theta = [z_{im}, b_m, \gamma_j | m = 1, \ldots, M - 1, \ i = 1, \ldots, n \ and \ j = 1, \ldots, p]' \ for \ an \ LK \quad (4-11a)$$

$$\Theta = [z_{im}, b_m, r_m, \gamma_j | m = 1, \ldots, M - 1, \ i = 1, \ldots, n \ and \ j = 1, \ldots, p]' \ for \ a \ GK \quad (4-11b)$$

We further define the parameter vector $\theta$ that only contains parameter $z$ and $b$. That is

$$\theta = [z_{im}, b_m | m = 1, \ldots, M - 1, \ i = 1, \ldots, n]' \quad (4-11c)$$

With the same argument as in the last section for the KIGP model, the likelihood term in (4-4) is always 1 for a KIGP model, regardless of the kernel type. Therefore the likelihood term is negligible.

As for the prior term, based on the prior assumption in Chapter 2, for an LK, we have:

$$\log(P(\Theta | H)) = \log(P(z, b | K, \gamma, H)) + \log(P(\gamma | H)) = \sum_{j=1}^{p} \{\gamma_j \log(\pi_j) + (1-\gamma_j)\log(1-\pi_j)\} - \frac{1}{2} \sum_{m=1}^{M-1} \{\log(\det(K_{\gamma_m} + I_n)) + (z_m - b_m 1_n)'(K_{\gamma_m} + I_n)^{-1}(z_m - b_m 1_n)\}$$

$$- \frac{1}{2} \sum_{m=1}^{M-1} \{\log(\det(K_{\gamma_m} + I_n)) + (z_m - b_m 1_n)'(K_{\gamma_m} + I_n)^{-1}(z_m - b_m 1_n)\}$$

$$\sum_{j=1}^{p} \{\gamma_j \log(\pi_j) + (1-\gamma_j)\log(1-\pi_j)\} - (M-1)[\alpha \log(\beta) - \log(\Gamma(\alpha))] + \sum_{m=1}^{M-1} \{(\alpha - 1)\log(r_m^2) - \frac{\beta}{r_m^2}\} \quad (4-12b)$$

For a GK, it is

$$\log(P(\Theta | H)) = \log(P(z, b | K_{\gamma}(r), H)) + \log(P(\gamma | H)) + \log(P(r | H))$$

$$= \sum_{m=1}^{M-1} \{\log(\det(K_{\gamma_m} + I_n)) + (z_m - b_m 1_n)'(K_{\gamma_m} + I_n)^{-1}(z_m - b_m 1_n)\} + \sum_{j=1}^{p} \{\gamma_j \log(\pi_j) + (1-\gamma_j)\log(1-\pi_j)\} + (M-1)[\alpha \log(\beta) - \log(\Gamma(\alpha))] + \sum_{m=1}^{M-1} \{(\alpha - 1)\log(r_m^2) - \frac{\beta}{r_m^2}\}$$
In (4-12), $K_{nm}$ is the same as defined in (2-24); $\pi_j$ is defined in (2-4a). A non-informative set for $\alpha$ and $\beta$ is not appropriate because otherwise the associated normalization term will lead the algorithm always reject the GK ([25, 44]). Hence, throughout this study, we always set $\alpha = \beta = 1$ as the semi-informative prior for any scale kernel parameter.

To calculate the posterior part in (4-4) for a KIGP_RJ_2 algorithm, we adopt the same approach used in the last section. That is, introducing a pilot period and using the yielded samples within pilot period to give an initial estimation for the log-posterior term in (4-4). Referring to the approach to approximating the log-posterior term (through 4-6, 4-8 and 4-9) for the KIGP_RJ_1 algorithm, we can find that, if the kernel function and the gene-selection vector are fixed, all these procedures and formulas can be well applied to the algorithm in this subsection without any modification. With this hint, we split the log-posterior term for an LK into 2 separate pieces by using the chain law:

$$\log(P(\Theta | LK, y)) = \log(P(z, b | K_y, LK, y)) + \log(P(\gamma | LK, y))$$

(4-13a)

For a GK, similarly it is:

$$\log(P(\Theta | GK, y)) = \log(P(z, b | K_y, r), GK, y)) + \log(P(\gamma, r | GK, y))$$

(4-13b)

To evaluate the posterior probabilities in (4-13a) and (4-13b), the term $\log(P(z, b | K_y, LK, y))$ and $\log(P(z, b | K_y, (r), GK, y))$ will be given by (4-15); the term $\log(P(\gamma | LK, y))$ can be estimated by (4-17); and the term $\log(P(\gamma, r | GK, y))$ can be evaluated by (4-18c). The detail derivations and discussions are provided in the following context.
First of all, with a given set of \( \gamma \) for an LK (or \( \gamma \) and \( r \) for a GK), the kernel matrix \( K_\gamma \) in (4-13) is fixed. This then is the same set for the KIGP_RJ_1 algorithm. Therefore we can evaluate \( \log(P(z,b \mid K_\gamma,H,y)) \) by using the similar approach in the last subsection. With the definition in (4-11c), we first denote \( \log(P(z,b \mid K_\gamma,H,y)) \) by \( \log(P(\theta \mid K_\gamma,H,y)) \).

In order to evaluate \( \log(P(\theta \mid K_\gamma,H,y)) \), we further define \( \theta^* \) as the adjusted set of \( \theta \):

\[
\theta^* = [z_{im}^*, b_m^* \mid m = 1,...,M - 1, i = 1,...,n]^T, \text{ where } b_m^* = b_m \text{ and }
\]

\[
z_{im}^* = \begin{cases} 
\log(-z_{im}) & \text{if } y_i = M \\
 z_{im} & \text{if } y_i = m, \text{ for } i = 1,...,n, \ m = 1,...,M - 1 \\
\log(z_{im}) & \text{otherwise}
\end{cases} \tag{4-14}
\]

We assume the conditional posterior distribution of \( \theta^* \), \( \log(P(\theta^* \mid K_\gamma,H,y)) \), can be well approximated by a multivariate Gaussian distribution. Therefore, we can adopt the on-line updating formula (4-8) to estimate the mean and variance for \( \theta^* \). With \( \theta^* \), the term \( \log(P(\theta \mid K_\gamma,H,y)) \) can therefore be approximated by

\[
\log(P(\theta \mid K_\gamma,H,y)) \approx -\frac{1}{2} \log(\det(\Sigma_{\theta^*})) - \frac{1}{2} (\theta^* - m_{\theta^*})' \Sigma_{\theta^*}^{-1} (\theta^* - m_{\theta^*}) \\
- \sum_{\forall i,y_i = M} \sum_{m=1}^{M-1} z_{im}^* - \sum_{\forall i,y_i = m} \sum_{m=1}^{M-1} z_{im}^* \tag{4-15}
\]

The following context discusses how to evaluate the other terms in (4-13). Let us first focus on the LK case. For (4-13a), we still need to approximate \( \log(P(\gamma \mid LK,y)) \).

Based the theory of Gibbs sampler, we know the output samples of \( \gamma \) from a KIGP simulation with an LK equivalently drawn from \( P(\gamma \mid LK,y) \). However, it is almost
impossible to estimate the joint posterior distribution for the vector $\gamma$ due to its extremely high dimension. Hence, an alternative is needed to give a functional estimation of the posterior of $\gamma$. Our proposed approach is to use the product of the marginal posterior distribution of each element of the gene-selection vector. If we look at the general odds ratio formula (4-4), we can see that the key is to properly estimate the ratio of the posteriors between two competing kernel types. With our alternative approximation, the kernel type using more significant genes will have a stronger Occam’s factor (odds between prior to posterior). Hence, the RJMCMC algorithm still would penalize the more complicated model, giving an edge to the simpler model.

To estimate the marginal posterior distribution for a gene, one only needs to divide the number of times this gene is selected during the simulation (after burn-in period) by the total number of iterations. However, since we prefer to provide a more consistent estimation after a pilot period, whereas the length of a pilot period is usually not long for the sake of the algorithm’s efficiency (because during a pilot period, the algorithm needs to run the KIGP simulation with each candidate kernel type). Note that within each iteration of a regular KIGP simulation in algorithm (2-30), we always need to estimate the conditional posterior distribution given all other parameters including all other elements of the gene-selection vector, therefore we suggest the use of another more robust strategy to estimate the marginal posterior distribution of a gene. That is, for the $j$-th element of the gene-selection vector, $\gamma_j$, the logarithm of its marginal posterior distribution mass function is estimated by

$$\log(P(\gamma_j \mid LK, y)) = \frac{1}{L} \sum_{l=1}^{L} \log(P(\gamma_j \mid LK, y, \gamma_j^{[l]}, \theta^{[l]})) \text{ for } j = 1, \ldots, p$$

(4-16)
In (4-16) symbol $\gamma_{-j}$ denotes the gene-selection vector after removing the element $\gamma_j$ and $\gamma_{[l]}$ is the $l$-th sample of $\gamma_{-j}$ after the burn-in period; $\theta$ is defined in (4-11c) and $\theta_{[l]}$ is the $l$-th sample of $\theta$; the value of $P(\gamma_j | LK, y, \gamma_{[l]}^{[l]}, \theta_{[l]})$ is estimated by (2-24) with an LK. In real practice, assuming the on-line estimation for $\log(P(\gamma_j | LK, y))$ after obtaining the $l$-th sample is denoted by $\log(P(\gamma_j | LK, y))_{[l]}$, we actually adopt the following deductive updating formula to estimate $\log(P(\gamma | LK, y))$ by:

$$
\log(P(\gamma | LK, y))_{[l+1]} = \sum_{j=1}^{L} \log(P(\gamma_j | LK, y))_{[l]} \quad \text{for } l = 0, 1, \ldots
$$

where

$$
\log(P(\gamma_j | LK, y))_{[l]} = \frac{l}{l+1} \log(P(\gamma_j | LK, y))_{[l]} + \frac{1}{l+1} \log(P(\gamma_j | LK, y, \gamma_{[-j]}^{[l]}, \theta_{[l]}))
$$

(4-17)

All the symbols in (4-17) are the same as those defined in (4-16).

For the simulation with a GK, we adopt the similar “product of marginal” way to approximate the term $\log(P(\gamma, r | GK, y))$. Comparing to the case with an LK, we need to further estimate $\log(P(\gamma | GK, y))$. Using the similar formula in (4-17), we first estimate $\log(P(\gamma | GK, y))$ in a real time fashion by:

$$
\log(P(\gamma | GK, y))_{[l+1]} = \sum_{j=1}^{L} \log(P(\gamma_j | GK, y))_{[l]} \quad \text{for } l = 0, 1, \ldots
$$

where

$$
\log(P(\gamma_j | GK, y))_{[l]} = \frac{l}{l+1} \log(P(\gamma_j | GK, y))_{[l]} + \frac{1}{l+1} \log(P(\gamma_j | GK, y, \gamma_{[-j]}^{[l]}, \theta_{[l]}, r_{[l]}))
$$

(4-18a)

In (4-18a), the symbols are the same as in (4-17) and the term $P(\gamma_j | GK, y, \gamma_{[-j]}^{[l]}, \theta_{[l]}, r_{[l]})$ is calculated via (2-24) with a GK (with parameter $r_{[l]}$).
For the width $r$ of a GK, as a scale parameter, again we first map them to its logarithm. In this study, we take logarithm with based 10 of $r$ as the target. Then, we try to appropriately estimate the posterior of $\log_{10} r$. Based on all the KIGP simulation results in this study, we find that posterior of $\log_{10} r$ could be very non-Gaussian in many cases. Therefore, it is not proper to use a simple Gaussian distribution fitting to estimate $P(\log_{10} r \mid GK, y)$. We adopt the similar empirical procedure as discussed in section 2.1.5.1 for a GK to fulfill this task. To make on-line updating for the estimation possible, we exclude the smoothing step therein. In more detail, we calculate the histogram of the logarithm of the sample values for the width with some proper number of bins. After the simulation, we divide the histogram counts within each bin by the total number of iteration multiplied by the length of bin, and the results are the estimation of the marginal posterior probability for logarithm of the width within the range of this bin. In practice, we suggest that $-2 \leq \log_{10}(r) < 2$ and that the bin length be set at 0.1. With this presumption, the on-line updating formula for the estimation of the marginal posterior distribution for the width of each classifier after obtaining $l+1$ samples of $r$ can be formulated as below:

$$\log(P(r_m \mid GK, y))^{[l+1]} \approx \log(f_{m,k}^{[l+1]}) - \log(0.1(l+1)) - \log_{10}(r_m)$$

with the histogram updating $f_{m,k}^{[l+1]} = f_{m,k}^{[l]} + 1$, for $m = 1, 2, ..., M - 1$, $l = 0, 1, ...$

where $k = \lceil (\log_{10}(r_m) + 2) / 0.1 \rceil$ \hspace{1cm} (4-18b)

In (4-18b), function $\lceil \bullet \rceil$ gives the closest integer for the inside value toward infinity. Referring to the general odds ratio formula (4-4), obviously, comparing KIGP with a
non-parametric kernel type such as LK, KIGP with a GK bearing an extra penalty due to its extra freedom with introducing a width parameter for each classifier in model. The approximation yielded in (4-18b) embodies this Occam factor penalty in a noticeable way. Summarizing (4-18a) and (4-18b), we have

\[
\log(P(\gamma, r \mid GK, y))^{[t+1]} \approx \sum_{j=1}^{p} \log(P(\gamma_j \mid GK, y))^{[t+1]} + \sum_{m=1}^{M} \log(P(r_m \mid GK, y))^{[t+1]}
\]

(4-18c)

The first term in the right hand side of (4-18c) is from (4-18a) and the second term is estimated by (4-18b). We thus can estimate the posterior term with GK in (4-4) after a burn-in period.

In summary, the KIGP_RJ_2 algorithm is similar to the KIGP_RJ_1 algorithm. The KIGP_RJ_2 algorithm that includes an LK and a GK as candidate kernel type is executed as follows:

The algorithm consists of 3 successive periods. In order, they are burn-in period, pilot period and main period. The algorithm consists of two consecutive phases, “before main period” (including burn-in period and pilot period) and “main period”.

**Phase of “before main period”:** During this phase, within each iteration, the algorithm is executed as follows:

1. For each of the candidate kernel types, run within their first phase of the KIGP algorithm in (2-30).

2. If the simulation is in the pilot period, we further execute the following steps:
   a. For each kernel type, sample set \( \theta \), including \( z \), \( b \) and.
b. Convert $\theta$ to $\theta^*$ via (4-14).

c. Update $m_{\theta^*}$ and $\Sigma_{\theta^*}$ via (4-8).

d. For the simulation with an LK, update $\log(P(\gamma_j | LK, y))$ for $j = 1, 2, ..., p$ via (4-17).

e. For the simulation with a GK, update $\log(P(\gamma_j | GK, y))$ for $j = 1, 2, ..., p$ with (4-18a) and $\log(P(r_m | GK, y))$ (actually the relative histogram $f_m$) for $m = 1, 2, ..., M - 1$ via (4-18b).

3. Return to step 1 until the required number of iterations is met.

(4-19a)

Phase of "main period": After the pilot period, the algorithm enters into the main period. In this phase, first randomly choose a kernel type as the currently accepted kernel type, say $H_k$, along with its associated parameter set $\Theta_k$. Set

$$LP_k = \log(P(\Theta_k | H_k)) - \log(P(\Theta_k | H_k, y)),$$

where if $H_k = LK$, $\log(P(\Theta_k | H_k))$ and $\log(P(\Theta_k | H_k, y))$ are respectively evaluated via (4-12a) and (4-13a); and if $H_k = GK$, $\log(P(\Theta_k | H_k))$ and $\log(P(\Theta_k | H_k, y))$ are respectively evaluated via (4-12b) and (4-13b). After the initialization step, within each iteration in the main period, the algorithm is executed as below:

1. Randomly (with equal probability) choose another kernel type $H_{k'}$, $k' \neq k$.

2. Keep on running within the first phase of the relevant KIGP algorithm in (2-30) under kernel type $H_{k'}$, yielding a sample of parameter set $\Theta_{k'}$.

3. Map $\theta_k$ to $\theta_{k'}$ via (4-14).
4. Update $m_{\theta_k}$ and $\Sigma_{\theta_k}$ via (4-8).

5. If $H_k = LK$
   a. Update $\log(P(\gamma_j \mid H_k, y))$ for $j = 1, 2, \ldots, p$ via (4-17).
   b. Calculate the log-prior $\log(P(\Theta_k \mid H_k))$ via (4-12a).
   c. Calculate the log-posterior $\log(P(\Theta_k \mid H_k, y))$ via (4-13a).

6. If $H_k = GK$
   a. Update $\log(P(\gamma_j \mid H_k, y))$ for $j = 1, 2, \ldots, p$ via (4-18a).
   b. Update $\log(P(r_m \mid GK, y))$ (actually the relative histogram $f_m$) for
      
      $m = 1, 2, \ldots, M - 1$ via (4-18b)
   c. Calculate the log-prior $\log(P(\Theta_k \mid H_k))$ via (4-12b).
   d. Calculate the log-posterior $\log(P(\Theta_k \mid H_k, y))$ via (4-13b).

7. Set $LP_k = \log(P(\Theta_k \mid H_k)) - \log(P(\Theta_k \mid H_k, y))$.

8. Calculate the odds ratio by $r = \exp(LP_k - LP_k)$ and accept the kernel type $H_k$
   with the probability $\min(1, r)$.

9. Return to step 1 until the required number of iterations is met.

10. After the RJMCMC simulation through step 1 to 9 stops, counting the
    frequency of acceptance for each kernel type and pick up the kernel type that
    appears most as the winning kernel type.

11. If the winner is LK, by using the samples drawn from the algorithm thus far to
    set the gene-selection vector via the procedure described in section 2.1.5.2.

12. If the winner is GK

153
a. Set the width parameter to the mode of its empirical posterior via the procedure described in subsection 2.1.5.1.

b. Keep on running the KIGP algorithm in (2-30) with this fitted GK in its "gene selection phase".

c. With the samples drawn from step b, set the gene-selection vector via the procedure described in subsection 2.1.5.2.

(4-19b)

The output of the KIGP_RJ_2 algorithm in (4-19) is a kernel type with fixed kernel function and a fixed gene-selection vector, which is a natural input for the KIGP_RJ_1 algorithm in (4-10). To be noted, rather than the relatively short burn-in period and pilot period, the major computation cost in the algorithm (4-19) happens within the regular KIGP simulation in step 2. The only main extra cost in the main phase is the calculation of the inverse of matrix $\Sigma_{\theta_0}$, with a complexity $O(n^3)$. Compared to the cost for a regular KIGP simulation, this is almost negligible. Therefore algorithm (4-19) entails the same amount of computation as running a KIGP with a regular kernel type, which significantly improves the efficiency for solving the kernel type competition problem in contrast to a rigorous CV procedure. This improvement can appear even more dramatic if the number of candidate kernel types is large, since the efficiency of the KIGP_RJ_2 algorithm is linearly proportional to that number.

It is very important to clarify one issue for implementing the algorithm (4-19). Although in (4-19) we only list LK and GK as the candidate kernel types, this algorithm in principle is applicable to any regular non-parametric kernel type and to any regular parametric kernel type with only a scale parameter. The term "regular kernel" here means
a kernel type whose function is pre-defined and does not depend on the training data. For example, NK proposed in Chapter 3 is based on training data and therefore is not a regular kernel.

Specifically, for a regular non-parametric kernel type, such as PK(1), PK(2), we can seamlessly embed it into the algorithm by simply running it with the procedure related to an LK in (4-19). As for a regular parametric kernel type that only contains a scale parameter, such as those defined in (4-20), we also can execute the KIGP_RJ_2 algorithm with it by running it with the procedure related to a GK. In order to enrich our regular kernel type list in this study, we define two more kernel functions as below:

Exponential kernel: \( K(x, x') = \exp\left(-\frac{||x-x'||}{r}\right) \)

where \( r > 0 \) is the width parameter  

(4-20a)

Manhattan kernel: \( K(x, x') = \exp\left(-\frac{||x-x'||_M}{r}\right) \)

where \( r > 0 \) is the width parameter  

(4-20b)

In (4-20a), the operator \( ||\cdot|| \) denotes the L-1 norm of the vector. In (4-20b), the operator \( ||\cdot||_M \) represents the Manhattan norm of a vector, which is defined as the summation of the absolute value of each element of the vector. Throughout this study, we shall refer to an exponential kernel with width \( r \) as \( EK(r) \) and refer to a Manhattan kernel with width \( r \) as \( MK(r) \). Both \( EK \) and \( MK \) only contain a scale parameter. Actually there are several other kernel types that contain only a scale parameter. To extend the algorithm in (4-19) to cover more regular parametric kernel type is virtually straightforward. Following the line of the derivation for the case with a GK in (4-19), there should be no obstacle to
complete the procedure with most regular kernels. However, reasonable priors and
deliberate design of proper procedure to estimate posteriors for the relevant kernel
parameters usually are necessary on this regard.

To demonstrate the efficiency of the KIGP_RJ_2 algorithm for the kernel
competition problem, in the simulation study of this chapter, besides LK and GK, we
shall also include PK(1), PK(2), EK and MK under consideration as well. Therefore, the
entry candidate list for the KIGP_RJ_2 algorithm in this chapter contains 6 kernel types:
LK, PK(1), PK(2), GK, EK and MK.

Fig. 4-1: Flow chart of the KIGP_RJ algorithm.
4.3 Conclusive Algorithm for KIGP Microarray Analysis and Simulation Results

With the two algorithms detailed in section 4.2, we can finally summarize the kernel-imbedded Gaussian Process framework developed in this study (Chapter 2 and Chapter 3) with RJMCMC kernel type competition for a multi-classification microarray analysis problem. We shall refer to this conclusive algorithm as the KIGP_RJ algorithm. The flow chart of this conclusive algorithm is plotted in Fig. 4-1 and its detailed description is as follows.

The input to a KIGP_RJ algorithm is simple: microarrays, their associated label and a list of kernel types of interest. Specially in this chapter, this list contains 6 regular kernels (including 3 non-parametric kernels LK, PK(1) and PK(2) and 3 parametric kernels GK, EK and MK) and 2 natural kernels (including NGFK and NGTK). Then the KIGP_RJ algorithm is executed as follows:

1. With a regular kernel type list as entry, run the KIGP_RJ_2 algorithm in (4-19), yielding a winning regular kernel type $H$ along with its fixed gene-selection vector. If the winner is a parametric kernel, the output also includes the fitted value for its associated kernel parameter.

2. Run the UR procedure given in (3-18), yielding a fixed gene-selection vector for candidate natural kernels.

3. With the output from step 1 and 2 and a kernel type list, including the winning regular kernel in step 1 and all investigated natural kernels, run the KIGP_RJ_1 algorithm, yielding the ultimate winning kernel type, along with all the associated model parameters and prediction for testing samples.
As a by-product of the KIGP RJ algorithm in (4-21), we shall obtain the conditional (regular kernel only) posterior probability for each regular kernel type after step 1 and the conditional posterior probability of the winning regular kernel along with the investigated natural kernels. These probabilities give us a hint on how powerful a kernel type is, comparatively, to the target microarray analysis problem. To be noted, the computation cost of this summary algorithm actually is not much over that of a regular KIGP simulation in (2-30).

We now illustrate a few examples to verify the correctness and effectiveness of algorithm (4-21). We first revisit the simulated examples designed in Chapter 2 for a multi-classification model (Example 1 and 2 in section 2.3.2.2) and then we apply the KIGP RJ algorithm to the 6 published datasets in Table 2-2. The details of experimental settings and procedures (including the number of iterations for burn-in period and main period) for all these examples can all be found in Chapter 2. Specifically for a KIGP RJ algorithm, we need an extra pilot period for each candidate kernel type. In the simulations for the two simulated examples, we used 500 iterations as the length of the pilot period when running the KIGP RJ 2 algorithm and use 100 iterations as the pilot period when running the KIGP RJ 1 algorithm. In the simulations for the six real datasets, we use 2000 iterations as the pilot period for each kernel type when running the KIGP RJ 2 algorithm and use 200 iterations as the pilot period when running the KIGP RJ 1 algorithm.

In the end, the posterior probabilities for the two simulated examples are listed in Table 4-1. For example 1 (linear case), interestingly, even the MR for the independent
test for KIGP with an LK is significantly worse than that for KIGP with a GK, the KIGP\_RJ\_2 algorithm still favored the LK in a noticeable way. We believe this is because the training samples for all classes are linearly separable, the algorithm therefore would pick the simpler LK. In example 2, KIGP with an MK marginally outperformed KIGP with a GK, showing MK may better fit a problem with sharp Bayesian decision functions. Overall, NGFK is the output winner for example 1 and MK (MK(1.02) for classifier 1 and MK(1.52) for classifier 2) is the output winner for example 2. The testing MR with an NGFK for example 1 is 0.089. The testing MR with an MK for example 2 is 0.053.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>KIGP_RJ_2</th>
<th>KIGP_RJ_1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LK</td>
<td>PK(1)</td>
</tr>
<tr>
<td>Ex. 1</td>
<td>0.28</td>
<td>0.27</td>
</tr>
<tr>
<td>Ex. 2</td>
<td>0.02</td>
<td>0.03</td>
</tr>
</tbody>
</table>

Table 4-1: Estimated posterior probabilities for the simulated examples via running the KIGP\_RJ algorithm. The columns labeled title “KIGP\_RJ\_2” gives the results for the simulation of the KIGP\_RJ\_2 algorithm; whereas the columns associated with label “KIGP\_RJ\_1” shows the results for the simulation with the KIGP\_RJ\_1 algorithm. The title WRK represents the winning kernel of the KIGP\_RJ\_2 simulation. For example 1, it is LK. For example 2, it is MK.

The posterior probabilities for the six real datasets are provided in Table 4-2. One can see that all the results are fairly consistent with what we obtained in chapter 2 and 3. Obviously, the leukemia dataset, the SRBCT dataset and the brain tumor dataset are very fit for a liner method. For the lymphoma dataset and the breast cancer dataset, all the kernel types (except PK(2)) perform closely, although a radius base kernel, such as GK,
EK and MK, might better fit for the nature of these two datasets. For the colon dataset, probably due to those possibly mislabeled training samples, the winning regular kernel chosen by the KIGP_RJ_2 algorithm was significantly outperformed by NGFK and NTFK.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>KIGP_RJ_2</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LK</td>
<td>PK(1)</td>
<td>PK(2)</td>
<td>GK</td>
<td>EK</td>
<td>MK</td>
<td>WRK</td>
<td>NGFK</td>
</tr>
<tr>
<td>Leukemia</td>
<td>0.38</td>
<td>0.34</td>
<td>0.01</td>
<td>0.09</td>
<td>0.08</td>
<td>0.10</td>
<td>0.89</td>
<td>0.07</td>
</tr>
<tr>
<td>SRBCT</td>
<td>0.36</td>
<td>0.39</td>
<td>0.01</td>
<td>0.08</td>
<td>0.08</td>
<td>0.08</td>
<td>0.42</td>
<td>0.33</td>
</tr>
<tr>
<td>Colon</td>
<td>0.16</td>
<td>0.19</td>
<td>0.05</td>
<td>0.21</td>
<td>0.19</td>
<td>0.20</td>
<td>0.16</td>
<td>0.43</td>
</tr>
<tr>
<td>Lymphoma</td>
<td>0.18</td>
<td>0.19</td>
<td>0.03</td>
<td>0.22</td>
<td>0.20</td>
<td>0.18</td>
<td>0.42</td>
<td>0.29</td>
</tr>
<tr>
<td>Breast Cancer</td>
<td>0.17</td>
<td>0.17</td>
<td>0.02</td>
<td>0.22</td>
<td>0.20</td>
<td>0.22</td>
<td>0.39</td>
<td>0.31</td>
</tr>
<tr>
<td>Brain Tumor</td>
<td>0.35</td>
<td>0.34</td>
<td>0.01</td>
<td>0.09</td>
<td>0.10</td>
<td>0.11</td>
<td>0.77</td>
<td>0.11</td>
</tr>
</tbody>
</table>

Table 4-2: Estimated posterior probabilities for the real datasets via running the KIGP_RJ algorithm. The captions for this table is same as in Table 4-2. For WRK, for Leukemia, it’s LK; for SRBCT, it’s PK(1); for Colon, it’s GK; for Lymphoma, it’s GK; for Breast Cancer, it’s GK (or MK); for Brain Tumor, it’s LK.

The simulation results in this section provide the evidence on the consistency of KIGP_RJ algorithm. However, the more important feature that the KIGP_RJ algorithm brings to the KIGP framework in this study is that it significantly improves the efficiency of selecting the best kernel type comparing to adopt a rigorous CV procedure. This is particularly significant for the regular kernel type competition without a sufficient number of independent testing samples. For example, if we want to find the best kernel type out of the 6 regular kernel types investigated in this chapter, for the colon dataset, as
we suggested in the context therein, we need to run 5 times 3-fold CV for each kernel type. As the result, we may need to run about $5 \times 3 \times 6 = 90$ regular KIGP (with 2/3 training sample size though) simulations to get the answer. Considering that the computation complexity of a KIGP is proportional to $O(n^3)$ and that the extra cost in the pilot period for a KIGP_RJ algorithm, under the current experimental setting the overall computation of the CV approach is about 20 times higher than that of running the KIGP_RJ algorithm.
Chapter 5

Summary and Discussions

This study was motivated by the data analysis challenges posed by microarray gene expression experiments and the mathematical optimality of the kernel-induced approach due to their ability to solve a non-linear classification problem in a feature space rather than in an observation space. With a probit regression setting and by assuming the link function between significant gene expression data and latent variable for the response label is a Gaussian process, a kernel-induced hierarchical Bayesian framework is developed for cancer classification problems by using microarray gene expression data. In the end, we successfully design a fully automated algorithm to solve this model and satisfactory results have been achieved.

All the simulation examples in this study demonstrate that, even without knowledge of the underlying generative model, the method proposed in this study performs almost always close to the theoretical Bayesian bound not only in cases with linear Bayesian classifiers, but also in cases with very non-linear Bayesian classifiers. Even with mislabeled training samples, this method still works reasonably well and more robustly than the other referred classic methods. This sheds light on its broader usability to microarray data analysis problems, especially to those for which linear methods work flakily.
Six published microarray datasets are as analyzed, and the results show that our proposed method makes perfect prediction for 5 of them with only using the least (or tied least) number of significant genes among all other referred state-of-the-art methods such as SVM, random forest, PLR, GP_ARD or PAM. For the only non-trivial dataset examined in this study, the colon dataset, under a rigorous multiple 3-fold CV test, our proposed method significantly outperforms all other referred benchmark microarray analysis methods, showing its superiority when analyzing a noisy dataset. Furthermore, with each microarray dataset, there is an identified significant gene set associated with our method. Due to its optimum prediction performance, this set of gene deserves a deep biological inspection for retrieving the possible cause of the target cancer type in a genome level.

A brief description of the fulfilled tasks of this study is summarized as follows. In Chapter 2, via adopting a variable selection approach, we develop an algorithm that is named by kernel-imbedded Gaussian Process (KIGP). The core of the KIGP algorithm is a Gibbs sampler. In detail, in section 2.1, by using pre-defined regular kernel types, we develop an algorithm with a cascading structure to retrieve the set of significant genes and accordingly to make class predictions for the testing set. If the used kernel type is parametric, KIGP also provides a fitted value for the kernel parameters. We especially developed the kernel parameter-fitting scheme for GK and it can be applied to any other kernel type that only contains a scale parameter. Under the multinomial probit regression model, the extension of the KIGP algorithm to a multi-classification problem is given in section 2.2.
Through a feature projection approach, in Chapter 3, we present another approach, natural kernel-imbedded Gaussian Process (NKIGP), to a microarray classification problem. Specifically, by assuming that significant gene expression data can be well modeled by a multivariate Gaussian distribution, under the natural kernel framework, we propose a data-driven kernel building procedure, named by natural Gaussian Fisher kernel (NGFK), for the KIGP framework. By assuming that the significant gene data after an orthogonal projection is of a multiple-univariate Student-t distribution, we further suggest another new natural kernel, natural Student-t Fisher kernel (NTFK). Since the NK building procedure introduced in this study heavily relies on the training data, we apply a univariate ranking procedure to make the significant gene selection. The NKIGP method turns out to be very powerful.

In Chapter 4, through a RJMCMC algorithm, we realize the simultaneous kernel type competition under a unifying Bayesian framework. With embedding the kernel type selection function into the KIGP framework, we create the connection between the two approaches in Chapter 2 and Chapter 3. As the result, at the end of Chapter 4, we come up with the conclusive algorithm of this study, "KIGP_RJ" algorithm in (4-21). The flow chart is drawn in Fig. 4-1. Through this algorithm, we can efficiently select the best kernel type from all the kernel types discussed in this study. The simulation results provide the evidence of the correctness and effectiveness of the KIGP_RJ algorithm. The output of a KIGP_RJ algorithm also includes a gene-selection vector along with other relative parameters associated with the winning kernel type.

The flow chart of the KIGP_RJ algorithm is summarized in Fig. 4-1, in which with a training set and a candidate kernel type list as the input, the KIGP_RJ algorithm
would output a kernel function with fixed kernel parameter. This kernel function is supposed to have the best performance based on the training set. The output of the algorithm also includes the associated model parameters and predictions for the testing test if applicable. As a comparison, we would like to revisit Fig. 1-9, in which we draw a conceptual framework for this study, which was envisioned by us during the very start of this project. Initially, we attempted to build a GP-based learner for microarray analysis, whose output should include a proper kernel function and a set of significant genes. Clearly, the KIGP_RJ algorithm shown in Fig. 4-1 realizes this virtual picture in Fig. 1-9. The KIGP_RJ algorithm delivers a satisfactory performance for all the datasets tested in this study, meeting our goal when we started this project.

The only disadvantage of the proposed KIGP framework in a real world application is its computational complexity. With the KIGP_RJ algorithm, we have already dramatically improved the overall efficiency of using the KIGP framework. Even so, however, the computation cost is still high. Considering the nature of an MCMC algorithm and the general way to solve a GP-based model, this actually is the way it should be. Nonetheless, the high computational cost of using a KIGP can be noticeably alleviated by taking a tradeoff to apply a prescreening procedure for gene selection. As discussed in Chapter 2, we can safely eliminate most genes that are with a low UR score and only apply the KIGP_RJ algorithm to the data for the top, say 200, genes. This alternative can further dramatically decrease the cost without losing much predictive performance. In practice, if computational capability is a serious concern, we recommend to adopt this simplified version of a KIGP.
There are still a few interesting problems left for future research. For example, it might be interesting to develop a SVM-based Bayesian learning method for microarray analysis. This method could help deal with applications with large sample size, for which a KIGP method might not be able to work efficiently. Another interesting problem would be building a parallel implementation of a KIGP algorithm. KIGP obviously always involves intensive computation, building the KIGP algorithm with parallelism therefore is definitely preferable since this could spread the working load of a KIGP across multiple processors. This actually is an important problem for real implementations of this study. Last but not the least, in the KIGP model, we assume the additive noise is Gaussian distributed, which implies an L-2 norm cost function, which is known as being more sensitive to outlier samples than an L-1 norm cost function. Therefore, it would be beneficial to develop a model with double exponentially distributed noise, such as a logistic regression based model. All the problems listed here are very interesting but are beyond the scope of this study.
Annex: Supplementary Documents

A. Descriptions of the selected Genes for the Real Datasets

The detail descriptions of the identified genes by KIGP for the four of the real dataset are listed in the following tables.

<table>
<thead>
<tr>
<th>Index</th>
<th>Accession #</th>
<th>Gene Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4847</td>
<td>X95735</td>
<td>Zyxin</td>
</tr>
<tr>
<td>3320</td>
<td>U50136</td>
<td>Leukotriene C4 synthase (LTC4S) gene</td>
</tr>
<tr>
<td>2020</td>
<td>M55150</td>
<td>FAH Fumarylacetoacetate</td>
</tr>
<tr>
<td>5039</td>
<td>Y12670</td>
<td>LEPR Leptin receptor</td>
</tr>
<tr>
<td>1834</td>
<td>M23197</td>
<td>CD33 CD33 antigen (differentiation antigen)</td>
</tr>
<tr>
<td>4499*</td>
<td>X70297</td>
<td>CHRNA7 Cholinergic receptor, nicotinic, alpha polypeptide 7</td>
</tr>
<tr>
<td>1745</td>
<td>M16038</td>
<td>LYN V-yes-1 Yamaguchi sarcoma viral related oncogene homolog</td>
</tr>
<tr>
<td>3847</td>
<td>U82759</td>
<td>GB DEF = Homeodomain protein HoxA9 mRNA</td>
</tr>
<tr>
<td>4196</td>
<td>X17042</td>
<td>PRG1 Proteoglycan 1, secretory granule</td>
</tr>
<tr>
<td>1779*</td>
<td>M19507</td>
<td>MPO Myeloperoxidase</td>
</tr>
<tr>
<td>6539</td>
<td>X65116</td>
<td>Epb72 gene exon 1</td>
</tr>
<tr>
<td>6376</td>
<td>M63652</td>
<td>PFC Properdin P factor, complement</td>
</tr>
<tr>
<td>3258</td>
<td>U46751</td>
<td>Phosphotyrosine independent ligand p62 for the Lck SH2 domain mRNA</td>
</tr>
<tr>
<td>2111</td>
<td>M62762</td>
<td>ATP6C Vacuolar H+ ATPase proton channel subunit</td>
</tr>
<tr>
<td>1882</td>
<td>M27891</td>
<td>CST3 Cystatin C (amyloid angiopathy and cerebral hemorrhage)</td>
</tr>
<tr>
<td>1829*</td>
<td>M22960</td>
<td>PPGB Protective protein for beta-galactosidase (galactosialidosis)</td>
</tr>
<tr>
<td>1249</td>
<td>L08248</td>
<td>INDUCED MYELOID LEUKEMIA CELL DIFFERENTIATION PROTEIN MCL1</td>
</tr>
<tr>
<td>2121</td>
<td>M63138</td>
<td>CTSD Cathepsin D (lysosomal aspartyl protease)</td>
</tr>
<tr>
<td>2288</td>
<td>M64526</td>
<td>DF D component of complement (adipsin)</td>
</tr>
<tr>
<td>1924*</td>
<td>M31156</td>
<td>PTKAR2B Protein kinase, cAMP-dependent, regulatory, type II, beta</td>
</tr>
</tbody>
</table>

Table A-1: Gene description of the genes selected by KIGP/LK for the leukemia dataset.
The Asterisks mark the genes that were not reported by the original publication [24].
<table>
<thead>
<tr>
<th>Index</th>
<th>Accession #</th>
<th>Gene Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>377</td>
<td>Z50763</td>
<td>H.sapiens mRNA for GCAP-II/uroguanylin</td>
</tr>
<tr>
<td>493</td>
<td>R87126</td>
<td>MYOSIN HEAVY CHAIN, NONMUSCLE (Gallus gallus)</td>
</tr>
<tr>
<td>249</td>
<td>M63391</td>
<td>Human desmin gene, complete cds.</td>
</tr>
<tr>
<td>267</td>
<td>M76378</td>
<td>Human cysteine-rich protein (CRP) gene, exons 5 and 6.</td>
</tr>
<tr>
<td>513</td>
<td>M22382</td>
<td>MITOCHONDRIAL PROTEIN P1 PRECURSOR (HUMAN)</td>
</tr>
<tr>
<td>14</td>
<td>H20709</td>
<td>MYOSIN LIGHT CHAIN ALKALI, SMOOTH-MUSCLE ISOFORM (HUMAN);</td>
</tr>
</tbody>
</table>

Table A-3: Gene description of the genes selected by KIGP/GK for the colon dataset [4].

(Gene 245, 267 and 765 corresponds to the same gene)
<table>
<thead>
<tr>
<th>Index</th>
<th>Clone ID</th>
<th>Gene Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1443</td>
<td>566887</td>
<td>chromobox homolog 3 (Drosophila HP1 gamma)</td>
</tr>
<tr>
<td>585</td>
<td>293104</td>
<td>phytanoyl-CoA hydroxylase (Refsum disease)</td>
</tr>
<tr>
<td>3009</td>
<td>366647</td>
<td>butyrate response factor 1 (EGF-response factor 1)</td>
</tr>
<tr>
<td>1116</td>
<td>784830</td>
<td>D123 gene product</td>
</tr>
<tr>
<td>2423</td>
<td>26082</td>
<td>very low density lipoprotein receptor</td>
</tr>
<tr>
<td>420*</td>
<td>203347</td>
<td>clathrin, heavy polypeptide-like 2</td>
</tr>
</tbody>
</table>

Table A-4: Gene description of the genes selected by KIGP/GK for the breast cancer dataset. The Asterisk marks the gene that was not reported by the original publication [29].

B. Heatmaps of the Selected Genes for the Real Datasets

The heatmaps of the identified genes by KIGP for the six real datasets are provided as follows. The chosen kernels are described in Chapter 2.

Fig. A-1: Heatmap of the identified genes by KIGP/LK for the leukemia dataset. The arrow marks the column corresponding to the suspicious testing sample (#31).
Fig. A-2: Heatmap of the identified genes by K1GP/PK for the SRBCT dataset.

Fig. A-3: Heatmap of the identified genes by K1GP/GK for the colon dataset.
The dashed arrows mark the columns for the suspicious samples (sample 18, 20, 45, 49 and 56).
Fig. A-4: Heatmap of the identified genes by KIGP/GK for the lymphoma dataset.

Fig. A-5: Heatmap of the identified genes by KIGP/GK for the breast cancer dataset.
Fig. A-6: Heatmap of the identified genes by KIGP/L.K for the brain tumor dataset.

C. Link to the C++ Code and an Example Dataset

The link to the C++ code:

http://www2.hawaii.edu/~xinz/My%20HTML/Article/GPKernel.h
http://www2.hawaii.edu/~xinz/My%20HTML/Article/GPKernel.cpp
http://www2.hawaii.edu/~xinz/My%20HTML/Article/BasicFunction.cpp
http://www2.hawaii.edu/~xinz/My%20HTML/Article/GPPrediction.cpp

The link to the colon dataset:

http://www2.hawaii.edu/~xinz/My%20HTML/Article/Colon_Train.txt
Bibliography


[41] Lin, Y., (2002), Support Vector Machines and The Bayes Rule in Classification, *Data Mining and Knowledge Discovery*, 6, 259-275


[64] Schölkopf, B., Mika, S., Burges, C., Knirsch, P., Müller, K., Rätsch, G., Smola, A., (1999), Input Space vs. Feature Space in Kernel-Based Methods, IEEE Trans. on Neural Networks, 10, 1000-1017


178


