Complexity-Based Classification of Software Modules

Jian Han Wang

A Thesis

in

The Concordia Institute

for

Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Quality Systems Engineering) at

Concordia University

Montréal, Québec, Canada

July 2008

Abstract

Complexity-Based Classification of Software Modules

Jian Han Wang

Software plays a major role in many organizations. Organizational success depends partially on the quality of software used. In recent years, many researchers have recognized that statistical classification techniques are well-suited to develop software quality prediction models. Different statistical software quality models, using complexity metrics as early indicators of software quality, have been proposed in the past. At a high-level the problem of software categorization is to classify software modules into fault prone and non-fault prone. The focus of this thesis is two-fold. One is to study some selected classification techniques including unsupervised and supervised learning algorithms widely used for software categorization. The second emphasis is to explore a new unsupervised learning model, employing Bayesian and deterministic approaches. Besides, we evaluate and compare experimentally these approaches using a real data set. Our experimental results show that different algorithms lead to different statistically significant results.

Acknowledgements

I would like to express my appreciation to my advisor, Dr. Nizar Bouguila, great thanks for him to help me go through this difficult process. His persistent encouragement and unconditional support grant me confidence to accomplish this work.
I would also like to thank to all my colleagues, for the bonding and sharing our lives within two years.
Finally, I would also like to thank my parents living in China, your love and care always give me power to move forward.

Table of Contents

List of Tables vii

List of Figures viii

1 Introduction 1

2 Empirical Evaluation of Selected Algorithms for Complexity-Based Classification of Software Modules 4

2.1 Modules Representation Using Complexity Metrics 4

2.2 Statistical Methods 5

2.2.1 Principal Components Analysis 6

2.2.2 Discriminant Analysis 7

2.2.3 Multiple Linear Regression 9

2.2.4 Logistic Regression 10

2.2.5 Support Vector Machine 11

2.2.6 Finite Mixture Models 12

2.3 Experimental Results 15

2.3.1 The MIS Data Set, Metrics and the Experimental Methodology 15

2.3.2 Experimental Results and Analysis 17

3 Software Modules Categorization Through Likelihood and Bayesian Analysis of Finite Dirichlet Mixtures 27

3.1 Introduction 27

3.2 ML Estimation of a Dirichlet Mixture 30

3.2.1 Estimation of the ξ_1 Parameters 31

3.2.2 Estimation of the ξ_2 Parameters 32

3.3 Bayesian Estimation of a Dirichlet Mixture 34

3.3.1 Convergence 37

3.4 Selection of the Number of Clusters 38

3.5 Experimental Results 40

3.5.1 Synthetic Data 40

3.5.2 Real Application 40

4 Conclusion 45

List of References **47**

List of Tables

2.1 Principal Components Analysis for MIS. 17

2.2 Type I and Type II errors, and the accuracy rates using different approaches with and without PCA. 19

2.3 Type I and Type II errors by processing first two principal components. 20

2.4 Absolute-value results of resampled paired t test with 11 software complexity metrics without PCA pretreatment. 21

2.5 Absolute-value results of resampled paired t test with PCA pretreatment. 24

2.6 Absolute-value results of resampled paired t test with PCA pretreatment and using the first two significant principal components. 25

2.7 Absolute-value results of resampled paired t test with 11 software complexity metrics with and without PCA-pretreatment. 25

2.8 Absolute-value results of resampled paired t test with 11 and 2 software complexity metrics. 26

3.1 Parameters of the different generated data sets (n_j represents the number of the elements in cluster j). 41

3.2 Estimated parameters for the different generated data sets using both the likelihood and Bayesian approaches. $|\hat{\alpha}_j|^L$, $\hat{\mu}^L_{j1}$, $\hat{\mu}^L_{j2}$, $\hat{\mu}^L_{j3}$ and $\hat{p}(j)^L$ are the estimated parameters using the likelihood approach. $|\hat{\alpha}_j|^B$, $\hat{\mu}^B_{j1}$, $\hat{\mu}^B_{j2}$, $\hat{\mu}^B_{j3}$ and $\hat{p}(j)^B$ are the estimated parameters using the Bayesian approach. 42

3.3 Type I and Type II errors using both the deterministic and Bayesian approaches. . . . 44

3.4 Classification probabilities (probabilities to be in the non fault-prone class) of the misclassified modules causing type I errors. 44

List of Figures

2.1 The relationship between the metrics and number of CRs. 22
2.2 Two-dimensional plot of variable coefficients and scores. 23

3.1 The Dirichlet distribution for different parameters. (a) $|\alpha|$=14, $\mu_1 = 0.25$, $\mu_2 = 0.5$, $\mu_3 = 0.25$. (b) $|\alpha|$=21, $\mu_1 = 0.16$, $\mu_2 = 0.68$, $\mu_3 = 0.16$. (c) $|\alpha|$=28, $\mu_1 = 0.125$, $\mu_2 = 0.75$, $\mu_3 = 0.125$. 29
3.2 Mixture densities for the generated data sets 40
3.3 Number of clusters found for the different generated data sets using likelihood approach. (a) Data set 1, (b) Data set 2, (c) Data set 3, (d) Data set 4.Here X axis represents the number of clusters, and Y axis represents the MDL value. 41
3.4 Number of clusters found for the different generated data sets using the Bayesian approach. (a) Data set 1, (b) Data set 2, (c) Data set 3, (d) Data set 4. Here X axis represents the number of clusters, and Y axis represents the MDL value. 43
3.5 Plot of multiple potential scale reduction factor values. 43

CHAPTER 1

Introduction

With the increasing need of complex computer systems, the advance in hardware performance, the size and complexity of softwares used is inevitably growing rapidly. Thus, more and more energy and investigation are devoted to the software quality field to seek techniques that can accurately reflect software performance and reliability [1]. Software is composed of a large number of relatively independent units called modules which perform certain functions [2]. One way to test software quality is to determine the number of faults in each module. These faults may be related, for instance, to changes [1] happening while the software is executing [4] and are in general in a small portion [2] of the modules [6,7]. Most of the time, people are not concerned about the exact number of changes, rather than setting a threshold. If the number of faults (i.e defects in a program that can cause incorrect execution [2]) found in certain module exceeds this previously set criterion, it is regarded as fault-prone, otherwise non fault-prone [8,9]. For example, if a threshold of two faults is set, each module having two or more changes will be assigned to the fault-prone group and considered unstable and with high-risk.
A software prediction model is viewed as an empirical tool using a certain algorithm to forecast modules types (i.e fault-prone or non fault-prone) [10]. A key common characteristic of these prediction models is that they establish a relationship between the measures of modules attributes and the types [11]. The fundamental construction of the predictive models is based upon the faults and corresponding measures collected from past similar programme development and maintenance scenarios. When the model is built, we can determine the quality and reliability of new modules, if the measures of their attributes are in hand. The understanding of the modules through prediction models helps target high-risk modules which need priority attention, extensive testing, redesign and improvement in early life cycle [12],

[1] See [3] for a discussion about the types and classes of changes that may occur.
[2] According to the 80/20 rule, about 20 percent of a software system is responsible for 80 percent of its errors, costs and rework [5].

which is very valuable, cost-effective, and improve the efficiency of inspection efforts [13]. It is not acceptable to postpone the assurance of software quality until the product's release. For instance, in telecommunication or military systems [8–10, 14, 15], if faults are not early identified, but found later in operational phase, any slightly changed signal or message used to communicate will likely cause expensive consequences. In addition, delaying correction in testing and operational phase may result in higher cost. Conversely, knowing the troublesome modules in time will guide the designers to optimize the development process and allocate the efforts to the right modules in dire need of being enhanced [16]. For example, predicting the high-risk modules during the design phase allows designers to refine or restructure the system to reduce its complexity. And if those are identified in the implementation phase, the majority of the test resources will be assigned to which are most likely to cause quality problems. Thus, a software predictive model, which can categorize program modules into fault-prone or non fault-prone, not only locate the troublesome modules earlier, but also benefits the designers to effectively use the resources to the accurate ones, which have internal faults, with the utmost probability. In addition, these models may even be used to guide maintenance activities during the operations phase [17, 18].
Since software categorization plays a critical role in the software quality field, nowadays more and more modeling pattern recognition [19], statistical analysis, and machine learning [20] techniques are employed in building predictive models such as neural networks [21, 22], discriminant power [23], fuzzy classification [24], classification trees [5, 19, 25–27], regression trees [28], support vector machine, discriminant analysis and finite mixture models. All these techniques are employed to build predictive models and extract information from massive data, this process is called "learning". In this thesis, two types of learning algorithms are studied, supervised and unsupervised. Supervised learning, as the name indicates, needs training data sets which are provided to support the extraction of rules and pattern out, to generate a discriminant function by looking at the input and output of the learning data sets. Then, test data sets apply this discriminant function to map input observations to desired outputs, fault-prone or non fault-prone, to evaluate the generalized ability of the built predictive models. Normally, training and test data sets are randomly selected from an identified data set with observations and their labels, collected from previous projects. Despite the great interest in prediction models, only few studies have been devoted to compare and evaluate the different techniques used. Unsupervised learning algorithms only use the model observations, having no need of their corresponding labels. In this context, unsupervised learning algorithms deserve much more focus and development. However, there are only few unsupervised algorithms that have been used by the software engineering community.
This thesis is composed of four chapters. The introductory chapter, presents the recent situation in

software categorization domain. The second chapter begins with the description of several widely used methodologies applied in software categorization context, then in order to make an evaluation and comparison based on recognized indicators, we conduct an experiment by using real data set. Consequently, we establish a new unsupervised algorithm, called finite Dirichlet mixture model, examined by deterministic maximum likelihood and Bayesian approaches in the third chapter. In the last chapter, we summarize the achieved results and conclude this thesis.

CHAPTER 2

Empirical Evaluation of Selected Algorithms for Complexity-Based Classification of Software Modules

In this chapter, we will perform a survey of selected algorithms including unsupervised and supervised ones, for software categorization. Using a real data set, the classification results are compared and analyzed.

2.1 Modules Representation Using Complexity Metrics

The different classification approaches that we will describe in this thesis represent each software module using complexity metrics which have been developed to measure software quality and capture modules features [2, 4, 29–31]. Indeed, each module is considered to be a multidimensional vector in the complexity metrics space. These metrics are not only part of measurable [1] software attributes which can be gathered in the early life cycle of software design, but also are proven indicators which describe the software complexity and analyze its improvement [35, 36]. In many previous studies, it was observed that the software complexity is directly related to software quality and fault-correction activity [2, 37–40], which means, for instance, that fault counts and change counts are highly correlated. Thus, measures of software complexity are good indicators understanding and modeling the quality of software.

Software metrics are constructed by a variety of measures of program codes. Many product metrics

[1]See [32–34], for instance, for interesting discussions about measurement theory in software engineering.

and techniques to evaluate them have been proposed [41–43]. In particular, Lines of Code (LOC) is generally closely related to the number of faults found later when executing software. In addition to the Lines of Code, there are other well-known and widely used measures of product metrics. For instance, Halstead's software science [44] is an approach dedicated to build software complexity measures by identifying a set of basic elements describing the modules, such as operands and operators [45]. Operands refer to variables and constants, and operators indicate symbols or combination of symbols that affect the values of operands. The basic measures of this approach is based upon four scalar numbers derived directly from the module's source code: (1) the number of unique operators (2) the number of unique operands (3) the total number of operators (4) the total number of operands. Furthermore, the basic Halstead complexity measures are combined in a number of ways to produce additional measures, which are widely adopted as indicators in vast majority cases. Halstead's complex metrics are popularly employed in evaluating mainstream programming, such as Fortran and Pascal.

During the past decade, object-oriented approaches have been extensively used in software development environments. The conceptual and structural nature of these approaches, have created new challenges in the software quality field such as exploring new and special metrics [46, 47], and assessing software quality in object-oriented environments [48]. A well-known example is Chidamber and Kemerer's metrics suite proposed in [49] and widely studied and evaluated in the literature [46, 50].

The main reason that software complexity metrics are widely used, is that they can be collected in the very early software life cycle. Some of them are obtained directly from measuring the source codes and high-level design, and some are even taken from the software specifications. However, since part of the components of complexity metrics is the combination of some of the others, there are potential linear relationship within them [51, 52]. Besides, some of the metrics used may be redundant with marginal contribution [53]. Thus, it is necessary to explore the structure of observations to understand the mutual collinearity existing within the components [54]. In this thesis, we adopt principal components analysis technique to investigate the underlying relationship between every two predictors and process observations beforehand.

2.2 Statistical Methods

Different techniques have been proposed to develop a predictive relationship between software complexity metrics and the categorization of the modules into fault-prone and non fault-prone [2]. These

[2]Note that some studies have re-examined the analysis under the assumption that only two classes can be distinguished by considering a number of differentiable groups instead of two (See [55, 56], for instance).

predictive models are built generally from examples [20] using training data sets composed of labeled observations (i.e modules taken, for instance, from historical projects). Then, according to these built models new unlabeled modules can be identified as fault-prone or non fault-prone which allow software engineers to detect troublesome modules in the early life-cycle of a software product. Before building a quality prediction model, an important step generally implemented is validating [57] and analyzing the software metrics used, to examine the interrelationship among them, to reduce the dimensionality of the observations describing the modules and then simplifying the quantity of calculations. Note that the validated metrics can then be applied on multiple projects [58]. Principal Components Analysis (PCA) is the most used technique for this task and allows the extraction of the most relevant information brought by the used metrics. In the next section, we will introduce these most successful techniques in details, mainly centering around their usefulness in software quality prediction.

2.2.1 Principal Components Analysis

Principal Components Analysis (PCA) is a widely used exploratory multivariate technique [59]. Suppose we have a set of N modules $\mathcal{X} = (x_1, x_2, \ldots, x_N)$, where each module is represented by a d-dimensional vector, of complexity metrics $x_{il}, l = 1, \ldots, d$, $x_i = (x_{i1}, x_{i2}, x_{i3}, ..., x_{id}) \in \mathbb{R}^d$, $i = 1, \ldots, N$, where two or more metrics have high degree of linear correlation. This is called multicollinearity, and it is a major problem in many models such as regression analysis built on the basic assumption that selected variables are independent [29]. When multicollinearity exists among some metrics, the established statistical model become unstable, and coefficients parameters estimated by training data sets are very sensitive [15]. Besides, the model will not be robust enough to forecast response variables of new observations. A solution to this problem is the application of PCA to transform correlated metric data into orthogonal variables. As in practice, software complexity metrics are often found highly correlated to each other and are a linear combination of a small number of orthogonal metric domains [60], PCA has been applied in many works [7, 10, 15, 61–63].

PCA finds a linear transformation W^T which maps the d-dimensional metrics vectors space into a new space with lower dimension $d^{new} < d$. The d^{new}-dimensional vectors x_i^{new} are given by:

$$x_i^{new} = W^T x_i \tag{1}$$

With PCA we try to find the optimal projection E which maximizes the determinant of the scatter matrix $W^T \Sigma W$ of the new projected samples $\mathcal{X}^{new} = (x_1^{new}, \ldots, x_N^{new})$

$$E = \arg\max_{W} |W^T \Sigma W| \tag{2}$$

where Σ is the scatter matrix of the original data

$$\Sigma = \sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})^T \tag{3}$$

$\bar{x}$ is the mean vector of $\mathcal{X}$

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{4}$$

and $E = [E_1, \ldots, E_{d^{new}}]$ is composed of the d-dimensional eigenvectors of S corresponding to the d^{new} largest eigenvalues [64].

2.2.2 Discriminant Analysis

Discriminant Analysis technique [59] is applied when we attempt to build a predictive model of groups membership based upon observed characteristics of each observation (i.e module). In software categorization case, this technique generates a discriminant function which can classify software modules as either high or low risk according to the software complexity metrics [7–9, 15]. This discriminant function, generated from a set of observations of labeled modules, can then be applied to new observations with software measurements but unknown groups membership. There are several discriminant analysis models (i.e linear, non linear and logistic discriminant model) that can be chosen depending on the data type of predictive variables such as all quantitative, all qualitative or mixed [65].

Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) was used extensively by software engineering researchers to both assess software quality [10, 15] and evaluate software metrics [66, 67]. LDA assumes that the classes are linearly separable and follow homoscedastic gaussian distributions. Under this assumption, one can show that the optimal subspace where we can perform the classification is given by the vectors W which are the solution of the following generalized eigenvalue problem

$$\Sigma_b W = \lambda \Sigma_w W \tag{5}$$

where Σ_w is the within-class scatter matrix and given by

$$\Sigma_w = \sum_{j=1}^{M} \sum_{i=1}^{n_j} (x_i - \bar{x}_j)(x_i - \bar{x}_j)^T \tag{6}$$

where n_j is the number of vectors in class j and $\bar{x}_j$ is the mean of class j. Σ_b is the between-class scatter matrix and given by

$$\Sigma_b = \sum_{j=1}^{M} (\bar{x}_j - \bar{x})(\bar{x}_j - \bar{x})^T \tag{7}$$

where M is the total number of classes. The linear discriminant model generally used, to differentiate fault-prone from non fault-prone modules, is based on the following generalized squared distance:

$$D_j^2(x) = (x - \bar{x}_j)^T \Sigma_p^{-1} (x - \bar{x}_j) \tag{8}$$

where, $\bar{x}_j$ represents the mean vector of class $j \in \{1, 2\}$ and Σ_p is the so-called pooled covariance matrix given by:

$$\Sigma_p = \frac{\sum_{j=1}^{2} n_j \Sigma_j}{\sum_{j=1}^{2} n_j} \tag{9}$$

where Σ_j is the covariance matrix of class j and n_j represents the number of modules in class j. Thus, the posterior probability of membership of x in class j is :

$$p_j(x) = \frac{e^{-1/2} D_j^2(x)}{\sum_{j=1}^{2} e^{-1/2} D_j^2(x)} \tag{10}$$

According to the discriminant function given by the previous equation, a vector x is assigned to the class j yielding to the greater posterior probability. Despite its effectiveness, a major inconvenient of LDA is the Gaussian assumption which is not the best choice [61]. A solution to this problem is nonparametric discriminant analysis. Another major drawback of LDA is the linearity of the classification surface. To overcome this problem, SVMs can be used to offer both linear and non-linear flexible classification surfaces. Moreover, discriminant analysis is less appropriate, when many of the metrics are discrete and an alternative approach in this case is logistic regression [68].

Nonparametric Discriminant Analysis

Nonparametric Discriminant Analysis (NDA) does not make assumptions about the distribution of the data and was widely used for classification in the case of software quality modeling [7–9, 61, 69–71]. Let f_j be the multivariate probability density function representing class j. Nonparametric discriminant analysis is based on the empiric estimation of the densities f_j which gives an approximation $\hat{f}_j$ to it as the following:

$$\hat{f}_j(x_i|\lambda) = \frac{1}{n_j} \sum_{i=1}^{n_k} K_j(x_i|x_{jk}, \lambda) \tag{11}$$

where $K_j(x_i|x_{jk}, \lambda)$ is a multivariate normal kernel on vector x_i, with modes at x_{jk} which is a vector in class j, and given by

$$K_j(x_i|x_{jk}, \lambda) = (2\pi\lambda^2)^{-n_j/2}|\Sigma_j|^{-1/2} \exp((\frac{-1}{2\lambda^2})(x_i - x_{jk})^T \Sigma_j^{-1}(x_i - x_{jk})) \tag{12}$$

where λ is a smoothing parameter chosen by optimizing the misclassification rates of cross validation on the training data set [69]. Then, the classification is based, in the case of our problem, on the following rule

$$class(x_i) = \begin{cases} 1 & \text{if } \frac{\hat{f}_1(x_i)}{\hat{f}_2(x_i)} > \frac{n_2}{n_1} \\ 2 & \text{otherwise} \end{cases} \tag{13}$$

2.2.3 Multiple Linear Regression

Multiple linear regression [72] performs a summary of the relationship between the module types, fault prone or non-fault prone, and the software complexity metrics, which is represented as a multivariate linear regression model. Here the determined module type, so-called response variable or dependent variable, is denoted as Y, and the software complexity metrics, which are composed of independent indicators, are represented as x_i. Written mathematically, the standard multiple regression is,

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_i x_{id} + \epsilon_i \tag{14}$$

where β_i are the coefficient parameters, and ϵ_i are normally distributed random variables, called error terms on the assumption that mean equals to 0 and variance is unknown and constant. Some approaches are widely employed to estimate the parameters, such as least square estimations, least absolute value estimation, relative least squares and minimum relative error procedures [73]. Least square estimation is the most used method among them, and the estimated regression model parameters are yielded by minimizing $\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2$, where $\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + ... + \hat{\beta}_i x_{id}$, $\hat{Y}_i$ and $\hat{\beta}_i$ represent estimated values.

Multiple linear regression models are built from a set of potential large number of predictive terms, and a subset of significant independent terms should be determined to enter into the multiple regression models [74]. Some techniques are employed for adding or removing explanatory variables from the model: forward selection, backward elimination and stepwise regression, which are all iterative procedures. Forward selection starts with an empty subset in the model and add one explanatory variable (which most contribute to the model) at a time, continuing the iterations until reaching a certain stop criterion. On the contrary, backward elimination beginning with all the predictors, removes one of them (considered the most redundant) in every iterative procedure. Stepwise regression [74] can be referred as a forward selection with replacement. In each subsequent iterative step, the model is evaluated, using computed statistical significance, with or without a potential predictor to see if it contributes to the explanatory power of the model. After determining the most significant complexity metrics and estimating the model parameters, the linear combination of the predictors can be used to predict if the future modules are high-risk or not.

2.2.4 Logistic Regression

Logistic regression [75] was extensively used in software engineering for both metrics validation [46] and modules classification [12, 18, 68, 76–78]. It is a widely applied statistical modeling technique when the dependant (i.e response) variable has only two possible values which is the case in our studied problem (fault-prone vs. non fault-prone). The independent variables (software metrics in our case), however, may be categorical, discrete or continuous. The logistic regression model is given by the following form [75]

$$\ln\left(\frac{\pi(x_i)}{1-\pi(x_i)}\right) = \beta_0 + \beta_1 x_{i1} + ... + \beta_d x_{id} \tag{15}$$

where $\pi(x_i)$ is the probability of the event: *the module* x_i *is fault-prone*, and has the following multivariate exponential form

$$\pi(x_i) = \frac{\exp(\beta_0 + \beta_1 x_{i1} + ... + \beta_d x_{id})}{1 + \exp(\beta_0 + \beta_1 x_{i1} + ... + \beta_d x_{id})} \tag{16}$$

The ratio $\frac{\pi(x_i)}{1-\pi(x_i)}$ is usually interpreted as odds of occurrence, which compares the probability of the event *fault-prone* to the probability of the *non fault-prone* one. This odds ranges form zero to infinity, whereas its logarithm ranges from 0 to 1, and called the log odds or the logit. From Eq. 15, we can see that the $\frac{\pi(x_i)}{1-\pi(x_i)}$ has a linear relationship with x_i, and the parameters $\beta_1, \beta_2, ..., \beta_d$, so-called regression coefficients, embody the changes in the log odds. The estimation of these coefficients is in general based on the maximum likelihood approach and can be carried out with a wide variety of statistical software packages [75].

In practical applications, after the logistic regression model is set up, a threshold is experimentally designated to determine if the new modules are troublesome or not. For example, the threshold can be determined through a classification rule that minimize the expected cost of misclassification [18, 68, 69]:

$$class(x_i) = \begin{cases} 1: fault-prone & \text{if } \frac{\hat{\pi}(x_i)}{1-\hat{\pi}(x_i)} > \frac{C_I}{C_{II}} \frac{\pi_{nfp}}{\pi_{fp}} \\ 2: non\ fault-prone & \text{otherwise} \end{cases} \tag{17}$$

where C_I and C_{II} are respectively the cost of type I (a non-fault prone is classified as fault-prone) and type II (a fault-prone is classified as non fault-prone) misclassification; and π_{nfp} and π_{fp} represent the prior probabilities of non fault-prone and fault-prone, respectively.

2.2.5 Support Vector Machine

Support Vector Machine (SVM) [79] is a two-class classification method that has been used successfully in many applications dealing with data classification in general and software modules in particular [80]. In the following, we briefly summarize the theory of SVM. For two-class pattern recognition, we try to estimate a function $f : \mathbb{R}^d \rightarrow \{\pm 1\}$ using l training d-dimensional vectors x_i and class labels y_i,

$$(x_1, y_1), \ldots, (x_l, y_l) \in \mathbb{R}^d \times \{\pm 1\} \tag{18}$$

after the training the function f should be able to correctly classify new test vectors x into one of the two classes. Suppose that we have a hyperplane separating the first class (positive class) for the the second class (negative class). The idea behind SVM is to find the optimal hyperplane permitting a maximal margin of separation between the two classes and defined by

$$w.x + b = 0 \quad w \in \mathbb{R}^d, b \in \mathbb{R} \tag{19}$$

corresponding to decision function

$$f(x) = sign(w.x + b) \tag{20}$$

where b is the distance to the hyperplane from the origin and w is the normal of the hyperplane which can be estimated through the use of training data by solving a quadratic optimization problem [79]. w can be estimated by

$$w = \sum_{i=1}^{l} v_i x_i \tag{21}$$

where v_i are coefficient weights.

In general, classes are not linearly separable. In order to overcome this problem, SVM can be extended by introducing a kernel K to map the data into another dot product space F using a nonlinear map

$$\Phi : \mathbb{R}^d \rightarrow F \tag{22}$$

In this new space F, the classes will be linearly separable. The kernel K is given by

$$K(x, x_i) = (\Phi(x).\Phi(x_i)) \tag{23}$$

and measures the similarity between data vectors x and x_i. Then, the decision rule is

$$f(x) = sign\left(\sum_{i=1}^{l} v_i K(x_i, x) + b \right) \tag{24}$$

An important issue here is the choice of the kernel function and some well-known classic choices are

- Polynomial with degree *d*:

$$K(x_i, x) = (x_i^T x + 1)^d \tag{25}$$

- Radial basis function (RBF) with parameter σ:

$$K(x_i, x) = \exp\left(\frac{-\|x_i - x\|^2}{2\sigma^2}\right) \tag{26}$$

- Sigmoid with parameters κ and θ:

$$K(x_i, x) = \tanh(\kappa x_i^T x + \theta) \tag{27}$$

2.2.6 Finite Mixture Models

Finite mixture models are among the most applied and accepted statistical approaches [81]. Finite mixture models have several clear attractions: they have a solid grounding in the theory of probability and statistics, they are flexible enough to approximate any other statistical model and they are a natural choice when the data to model is heterogenous [81]. Moreover, finite mixtures permit a formal approach to unsupervised learning. The use of finite mixture models as a statistical tool for early prediction of fault-prone program modules has been investigated, for instance, in [82]. Finite mixtures can be viewed as a superimposition of a finite number of component densities and thus adequately model situations in which each data element is assumed to have been generated by one (unknown) component. More formally, a finite mixture model with M components is defined as

$$p(x_i|\Theta) = \sum_{j=1}^{M} p(x_i|\theta_j) p_j \tag{28}$$

The parameters of a mixture for M clusters are denoted by $\Theta = (\theta_1, \dots, \theta_M, P)$, where $P = (p_1, \cdots, p_M)$ is the mixing parameter vector. Of course, being probabilities, the p_j must satisfy $0 < p_j \leq 1, \quad j = 1, \dots, M$ and $\sum_{j=1}^{M} p_j = 1$. The choice of the component model $p(x_i|\theta_j)$ is very critical in mixture decomposition. The number of components required to model the mixture and the modeling capabilities are directly related to the component model used [81]. In the past two decades, much effort has been devoted to Gaussian mixture models estimation and selection (i.e determination of the number of components).

The multivariate Gaussian probability density function is the common assumption when using finite mixture models and is given by

$$p(x_i|\theta_j) = \frac{\exp[\frac{1}{2}(x_i - \mu_j)^T \Sigma_j^{-1}(x_i - \mu_j)]}{(2\pi)^{d/2}|\Sigma_j|^{1/2}} \tag{29}$$

where μ_j and Σ_j denote the mean and covariance metrix of each component respectively. Thus, in the case of a finite Gaussian mixture model, we have $\theta_j = (\mu_j, \Sigma_j)$.

An important problem in the case of finite mixture models is the estimation of the parameters. During the last two decades, the method of maximum likelihood (ML) has become the most common approach to this problem [81]. It is well known that the maximum likelihood (ML) estimate:

$$\hat{\Theta}_{ML} = \arg\max_{\Theta}\{L(\Theta, \mathcal{X})\} \tag{30}$$

where $L(\Theta, \mathcal{X})$ is the log-likelihood corresponding to a M-component is:

$$L(\Theta, \mathcal{X}) = \log \prod_{i=1}^{N} p(x_i|\Theta) = \sum_{i=1}^{N} \log \sum_{j=1}^{M} p(x_i|\theta_j)p_j \tag{31}$$

The maximization defining the ML estimates is subject to the constraints over the mixing parameters and can not be found analytically [81]. However, the ML estimates of the mixture parameters can be obtained using expectation maximization (EM) and related techniques [81]. The EM algorithm is a general approach to maximum likelihood in the presence of incomplete data. In EM, the "complete" data are considered to be $y_i = \{x_i, z_i\}$, where $z_i = (z_{i1}, \ldots, z_{iM})$, with:

$$z_{ij} = \begin{cases} 1 & \text{if } x_i \text{ belongs to class } j \\ 0 & \text{otherwise} \end{cases} \tag{32}$$

constituting the "missing" data. The relevant assumption is that the density of an observation x_i, given z_i, is given by $\prod_{j=1}^{M} p(x_i|\theta_j)^{z_{ij}}$. The resulting *complete-data log-likelihood* is:

$$L(\Theta, \mathcal{Z}, \mathcal{X}) = \sum_{i=1}^{N}\sum_{j=1}^{M} z_{ij} \log(p(x_i|\theta_j)p_j) \tag{33}$$

where $\mathcal{Z} = (z_1, \ldots, z_N)$. The EM algorithm produces a sequence of estimates $\{\Theta^t, t = 0, 1, 2\ldots\}$ by applying two steps in alternation until some convergence criterion is satisfied:

1. **E-step:** Compute $\hat{z}_{ij}$ given the parameter estimates from the initialization:

$$\hat{z}_{ij} = \frac{p(x_i|\theta_j)p_j}{\sum_{l=1}^{M} p(x_i|\theta_l)p_l}$$

2. **M-step:** Update the parameter estimates according to:

$$\hat{\Theta} = \arg\max_{\Theta} L(\Theta, \mathcal{Z}, \mathcal{X})$$

The quantity $\hat{z}_{ij}$ is the conditional expectation of z_{ij} given the observation x_i and parameter vector Θ. The value z^*_{ij} of $\hat{z}_{ij}$ at a maximum of equation (33) is the conditional probability that observation i belongs to class j (the *a posteriori* probability); the classification of an observation x_i is taken to be $\{k/z^*_{ik} = max_j z^*_{ij}\}$, which is the Bayes rule. When we maximize the function given by equation 9, we obtain:

$$p_j^{(t+1)} = \frac{1}{N}\sum_{i=1}^{N} \hat{z}_{ij} \tag{34}$$

$$\mu_j^{(t+1)} = \frac{\sum_{i=1}^{N} \hat{z}_{ij} x_i}{\sum_{i=1}^{N} \hat{z}_{ij}} \tag{35}$$

$$\Sigma_j^{(t+1)} = \frac{\sum_{i=1}^{N} \hat{z}_{ij}[(x_i - \mu_j^{(t)})(x_i - \mu_j^{(t)})^T]}{\sum_{i=1}^{N} \hat{z}_{ij}} \tag{36}$$

Another important problem now is the selection of the number of components M which best describes the data. For this purpose, many approaches have been suggested. From a computational point of view, these approaches can be classified into three classes: deterministic, stochastic, and resampling methods [81]. The most used approaches, however, are the deterministic methods which can themselves be classified in two main classes: in the first, we have approximate Bayesian criteria like the Schwarz's Bayesian information criterion (BIC) [83] and the Laplace-empirical criterion (LEC) [81]. The second class contains approaches based on information/coding theory concepts such as the minimum message length (MML) [84], Akaike's information criterion (AIC) [85], and minimum description length (MDL) criterion [86]. A more detailed survey of selection criteria approaches can be found in [81]. For instance, the authors in [82, 87], have used the AIC criterion given by

$$AIC(M) = -2L(\Theta, \mathcal{X}) + 2N_p \tag{37}$$

where N_p is the number of parameters in the model and is equal to $Md + (M-1) + Md(d+1)/2$ in the case of finite Gaussian mixture models. The selection of the optimal number of clusters M^* is done by $M^* = \arg\min_M AIC(M)$.

2.3 Experimental Results

In this section, we experimentally evaluate the performance of the different approaches presented in the previous section on a real data set called Medical Imaging System (MIS) [2]. In the following, we first describe the data set, the metrics used and the experimental methodology, then we give and analyze the experimental results.

2.3.1 The MIS Data Set, Metrics and the Experimental Methodology

MIS is a widely used commercial software system consisting of about 4500 routines written in approximate 400,000 lines of Pascal, FORTRAN, and PL/M assembly code. The practical number of changes (faults) as well as 11 software complexity metrics of each module in this program were determined during three-years system testing and maintenance. Basically, the MIS data set used in this thesis, is composed of 390 modules and each module is described by 11 complexity metrics acting as variables:

- LOC is the number of lines of code, including comments.
- CL is the number of lines of code, excluding comments.
- TChar is the number of characters
- TComm is the number of comments.
- MChar is the number of comment characters.
- DChar is the number of code characters
- $N = N_1 + N_2$ is the program length, where N_1 is the total number of operators and N_2 is the total number of operands.
- $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$ is an estimated program length, where η_1 is the number of unique operators and η_2 is the number of unique operands.
- $N_F = (\log_2 \eta_1)! + (\log_2 \eta_2)!$ is Jensen's [39] estimator of program length.
- V(G), McCabe's cyclomatic number, is one more than the number of decision nodes in the control flow graph.
- BW is Belady's bandwidth metric, where

$$BW = \frac{1}{n} \sum_i i L_i \tag{38}$$

  and L_i represents the number of nodes at level i in a nested control flow graph of n nodes [39]. This metric indicates the average level of nesting or width of the control flow graph representation of the program.

Figure 2.1 shows the number of faults found in the software as a function of the different complexity metrics. According to this figure, it is clear that the number of changes (or faults) increases as the metrics values increase.

In documented MIS data set, modules 1 to 114 are regarded as non fault-prone (number of faults less than 2), and those with 10 to 98 faults are considered to be fault-prone. Thus, there are 114 non fault-prone and 89 fault-prone modules.

Resampling is an often used technique to test classification algorithms by generating training and test sets. The training set is used to build the software quality prediction model, and the test set is used to validate the predictive accuracy of the model. In our experiments, we have used k-fold cross validation where original data sets are divided into k subsamples of approximately equal size. Each time one of the k subsamples is selected as test data set to validate the model, and the remaining $k-1$ subsamples acts as training data sets. Then, the process is repeated k times, with each of the k subsamples used exactly once as test data set. The k results are averaged to produce a misclassification error. Our specific resampling choice was 10-fold cross validation. In the case of our problem, there are two types of misclassification, type I and type II. Type I misclassification occurs when a non fault-prone module is wrongly classified as fault-prone and type II misclassification occurs when a fault-prone modules is mistakenly classified as non fault-prone. In our experiments, type I and type II misclassification rates are used as the measure of effectiveness and efficiency to compare the different selected classification algorithms. In order to assess the statistical significance of the different results achieved by supervised algorithms, we have used Student's t test; and for unsupervised one (i.e finite mixture model), a test for the difference of two proportions has been employed [88]. To conduct Student's t test, let $p_A^{(i)}$ be the misclassification rate of test data set i (i from 1 to 10) by algorithm A, and $p_B^{(i)}$ represents the same meaning. If we suppose 10 differences $p^{(i)} = p_A^{(i)} - p_B^{(i)}$ are achieved independently, then we can use Student's t test to compute the statistic $t = \bar{p}\sqrt{n}/\sqrt{\frac{\sum_{i=1}^{n}(p^{(i)}-\bar{p})^2}{n-1}}$, where n=10 and $\bar{p} = \frac{1}{n}\sum_{i=1}^{n} p^{(i)}$. Using the null hypothesis, this Student's distribution has 9 (n-1) degrees of freedom. In this case, the null hypothesis can be rejected if $|t| > t_{9,0.975} = 2.262$. To compare the results achieved by unsupervised algorithms, we adopt another statistical test to measure the difference. Let p_A represents the proportion of misclassified modules by algorithm A, so does p_B. Suppose p_A and p_B are normally distributed, so that their quantity of difference $(p^A - p^B)$ is normally distributed as well. The null hypothesis is rejected if $|z| = |(p^A - p^B)/\sqrt{2p(1-p)/n}| > Z_{0.975} = 1.96$, where $p = (p^A + p^B)/2$.

2.3.2 Experimental Results and Analysis

PCA Results

As a first step in our experiments, we have applied PCA to the MIS data set. Table 2.1 shows highest five eigenvalues as well as their corresponding eigenvectors and they express 98.57% of the features of the datasets in all. The columns from domain 1 to domain 5 are the principal component scores. According to this table, we can see that the first two largest eigenvalues express up to 90.8% information of the original dataset and then could be considered as comprehensive to some extent to describe the MIS dataset. Fig. 2.2 shows the PCA results by considering the first two components. Each of

Table 2.1: Principal Components Analysis for MIS.

Complexity Matrix	Domain 1	Domain 2	Domain 3	Domain 4	Domain 5
LOC	0.3205	0.0903	-0.0526	-0.2928	-0.4016
CL	0.3159	0.0270	0.1029	0.3481	-0.5452
TChar	0.3226	0.1287	-0.1794	0.1792	0.1572
TComm	0.2992	0.1577	-0.2484	-0.6689	-0.0144
MChar	0.2729	0.2911	-0.6850	0.2304	0.2915
DChar	0.3230	0.0191	0.2246	0.0319	0.0341
N	0.3176	0.0056	0.2785	-0.0172	0.0772
$\hat{N}$	0.3167	0.0092	0.3312	-0.0078	0.3617
NF	0.3166	0.0120	0.3358	0.0007	0.3589
V(G)	0.3052	-0.2011	-0.0763	-0.4917	-0.3771
BW	0.1751	-0.9077	-0.2593	0.1319	0.1502
Eigenvalue	9.1653	0.8224	0.4662	0.2330	0.1546
%Variance	83.32	7.48	4.24	2.12	1.41
%Cumulative	83.32	90.8	95.04	97.16	98.57

the eleven predictors is represented in this figure by a vector, the direction and length of which denote how much contribution to the two principal components the predictor provides. The first principal component, represented by the horizontal axis, has positive coefficients for all components. The second principal component, represented by the vertical axis, has positive coefficients for the components $BW', V(G)$, almost no coefficients for the components $DChar, N, \hat{N}, NF$, and negative coefficients for the remaining five. Note that in Fig. 2.2, the components $BW', V(G)$ and $MChar$ are standing out, which indicate that they have less correlation with other indicators. On the contrary, the indicators $DChar, N, \hat{N}$, and NF are highly correlated.

Classification Result

In this subsection, we present the results obtained using the different classification approaches that we have presented in the previous section. Table 2.2 shows these results with and without PCA pretreatment. In this table, type I and type II errors, and the accuracy rates which represent the ratio of the corrective classification modules to the total, are employed as the indicators to compare the overall classification capabilities. Comparing the different approaches using the accuracy rates, it is clear that the PCA pre-process improves generally the results and that LDA with PCA pretreatment performs best in our case, achieving highest accuracy rate 88.76%.

Table 2.3 lists the results achieved by using only the first two principal components as input to the selected algorithms except multiple linear regression (it is inappropriate to evaluate multiple linear regression by only two predictors). By comparing these results with the results shown in table 2.2, it is clear that in most of the cases, the results are better when we consider all principal components. The only exception is the results achieved by Gaussian finite mixture model. When tracking the intermediate variables, it occurs that, for each module, the two procedures with or without PCA pretreatment, respectively, arrive at the same probability of being fault-prone, as well as being non fault-prone. Table 2.3 also shows that, Logistic Regression with PCA performs best, orderly followed by NDA with PCA and LDA with PCA. SVM technique still functions here, but when classifying with Sigmoid kernel function, the accuracy rate decreases a lot.

Tables from 2.4 to 2.8 show the absolute value results of Student's t test when using different approaches with and without PCA. The statistical significance tests are conducted in order to make extensive comparisons under various circumstances. Tables 2.4 and 2.5 show comparisons between the different classification methods using the total eleven software complexity metrics with and without PCA, respectively. Table 2.6 shows also cross-comparisons, but with the first two significant principal components. In table 2.7 and 2.8, we investigate the statistical significance of the difference between the results achieved by each approach when we apply it with and without PCA by considering all the principal components and the first two most important components, respectively. The results in these four tables are computed using the outputs of every two algorithms, and any absolute value larger than $t_{9,0.975} = 2.262$ represents a statistical difference. The inspection of these two tables reveals, that on the one hand the disparity do exist between some of the algorithms, being wise to select simpler algorithm if the classification accuracy is not significantly different between the two; on the other hand we must point out that using merely the evaluation results with MIS data sets to measure the candidate

Table 2.2: Type I and Type II errors, and the accuracy rates using different approaches with and without PCA.

	Type I error	Type II error	Accuracy Rate
LDA	9.22%	16.47%	87.24%
LDA + PCA	6.92%	17.40%	88.76%
NDA	1.60%	30.79%	85.71%
NDA + PCA	2.43%	24.12%	88.17%
Logistic Regression	9.08%	18.51%	87.21%
Logistic Regression + PCA	6.43%	20.43%	88.14%
Multiple Linear Regression	22.37%	10.64%	82.69%
Multiple Linear Regression + PCA	12.04 %	13.08%	85.69%
SVM (Polynomial)	10.67%	28.11%	81.59%
SVM (Polynomial) + PCA	12.94%	24.61%	81.76%
SVM (RBF)	14.33	27.22	79.95%
SVM (RBF) + PCA	13.14%	20.55%	82.88%
SVM (Sigmoid)	30.25%	26.63%	70.88%
SVM (Sigmoid) + PCA	33.10%	25.31%	72.28%
Gaussian Mixture Model	1.75%	41.57%	80.78%
Gaussian Mixture Model + PCA	1.75%	41.57%	80.78%

algorithms is inappropriate to reach an absolute conclusion about the performance of the different approaches. Recent studies show that some factors seriously affect the performance of the classification algorithms [89]. Data set characteristics and training data set size are dominating factors. According to some empirical research,"best" prediction technique, depends on the context or data set characteristics. For example, generally LDA outperforms for data sets coming from Gaussian Distribution or with some outliers. Moreover, increasing the size of training data sets is always welcomed and improve the prediction results.

To sum up what we mentioned above, even if all the candidate classification algorithms have certain ability to partition data sets, choosing proper classifier strongly depends on the data sets characteristics and the comparative advantages of each classifier. LDA is more suitable for data sets following Gaussian distribution and with unequal within-class proportion. The essence of this algorithm is trying to find the linear combination of the predictors which most separate the two populations, by maximizing the between-class variance, and at same time minimizing the within-class variance. However, to analyze non-Gaussian data which are not linearly related or without common covariance within all groups, the logistic regression is preferred. But, logistic regression has its own underlying assumptions and inherent restrictions. In empirical applications, logistic regression is better for discrete outcomes. Besides, under the circumstances of continuous responses, multiple regression is more powerful. As logistic regression

Table 2.3: Type I and Type II errors by processing first two principal components.

	Type I error	Type II error	Accuracy Rate
LDA	1.77%	37.53%	82.81%
LDA + PCA	1.55%	36.37%	83.23%
NDA	2.25%	38.83%	82.85%
NDA + PCA	2.25%	38.00%	83.36%
Logistic Regression	5.63%	22.27%	86.67%
Logistic Regression + PCA	3.76%	24.48%	87.09%
SVM (Polynomial)	9.09%	88.17%	60.54%
SVM (Polynomial) + PCA	13.49%	23.63%	81.74%
SVM (RBF)	24.17%	18.23%	78.38%
SVM (RBF) + PCA	16.46%	19.09%	82.26%
SVM (Sigmoid)	71.73%	35.32%	45.28%
SVM (Sigmoid) + PCA	51.27%	31.69%	58.24%
Gaussian Mixture Model	20.17%	15.73%	81.77%
Gaussian Mixture Model + PCA	20.17%	15.73%	81.77%

NDA has no requirement concerning the distribution of data. Multiple Linear Regression is very effective when dealing with small quantity of independent variables, but easily being affected by outliers, so identifying and removing these outliers before building the model is necessary. The linear regression model generates unique coefficient parameters to every predictor. So, if the undesired outliers exist in training data sets, the built model with these fixed parameters can not provide reliable prediction for the software modules to test. In this context, other methods robust to the contamination of data (e.g. robust statistics) should be used. Regarding SVM, it transfers data sets into another high-dimensional feature space by means of kernel function, and finds support vectors as the determinant boundary to separate data set. Because the classification is achieved by maximizing the margin between the two classes, that process maximizes the generalization ability of this learning machine, which will not be deteriorated even if the data are somewhat changed within their original range. However, a drawback of SVM is that the kernel function computation is time-consuming. Finite mixture model is an unsupervised statistical approach which permits the partition of data without the training procedure. This technique should be favored when historical data is very costly or hard to collect.

Table 2.4: Absolute-value results of resampled paired t test with 11 software complexity metrics without PCA pretreatment.

	t test (type I error)	t test (type II error)
	vs. LDA	vs. LDA
NDA	2.0775	2.0360
Logistic Regression	0.0358	0.2456
Multiple Linear Regression	2.0962	1.1881
SVM (Polynomial)	0.3233	1.3729
SVM (RBF)	1.6156	1.1884
SVM (Sigmoid)	3.8010	1.2808
	vs. NDA	vs. NDA
Logistic Regression	1.9266	5.7418
Multiple Linear Regression	3.9220	3.7158
SVM (Polynomial)	3.2742	0.4011
SVM (RBF)	3.8099	0.8718
SVM (Sigmoid)	4.9427	0.6865
	vs. Logistic Regression	vs. Logistic Regression
Multiple Linear Regression	1.7345	1.1734
SVM (Polynomial)	0.4001	1.1290
SVM (RBF)	1.2063	2.1146
SVM (Sigmoid)	4.3904	1.1760
	vs. Multiple Linear Regression	vs. Multiple Linear Regression
SVM (Polynomial)	1.7914	2.7224
SVM (RBF)	1.3103	2.5823
SVM (Sigmoid)	0.8530	2.4620
	vs. SVM (Polynomial)	vs. SVM (Polynomial)
SVM (RBF)	0.7176	0.1114
SVM (Sigmoid)	3.6469	0.2349
	vs. SVM (RBF)	vs. SVM (RBF)
SVM (Sigmoid)	2.4779	0.0879

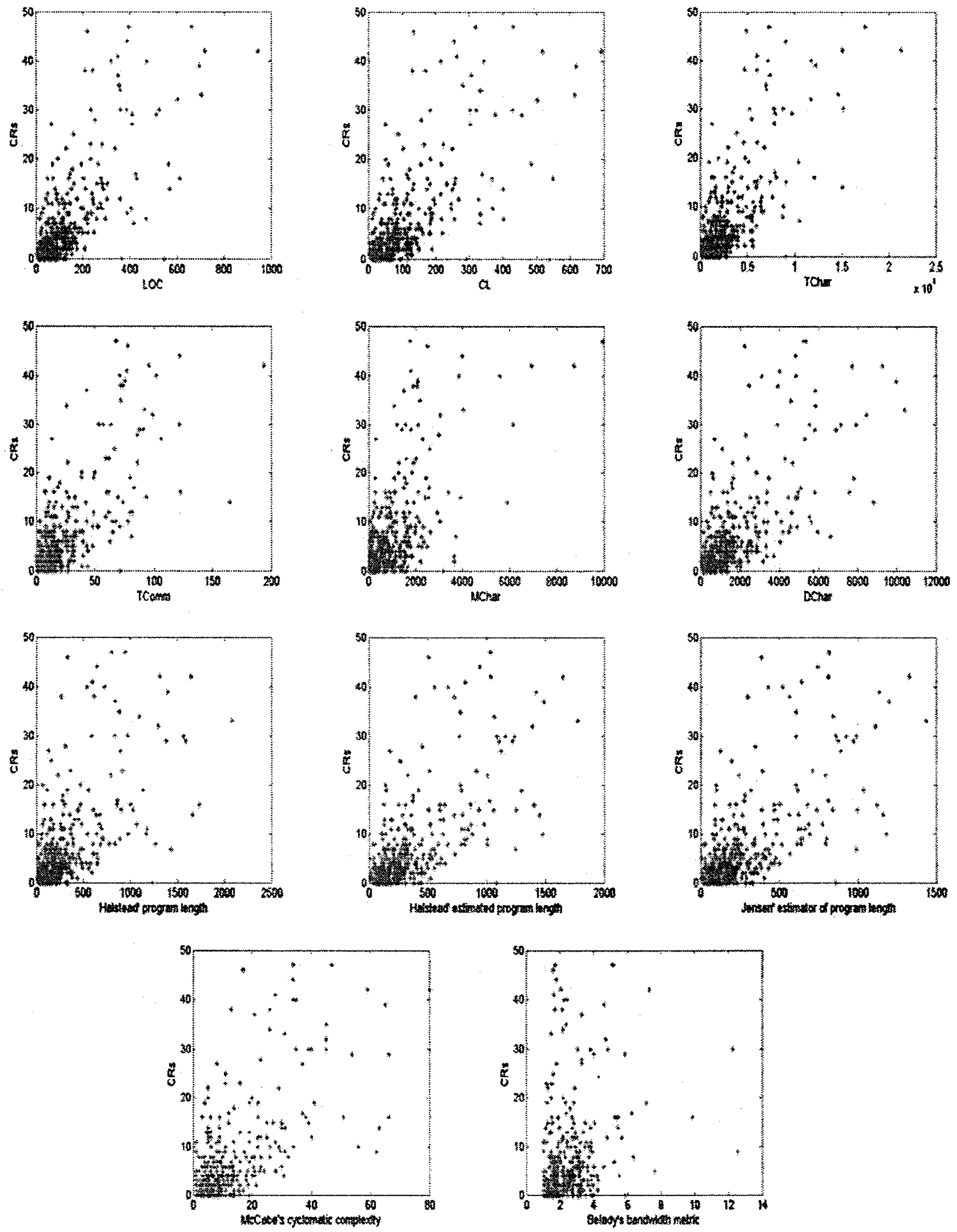

Figure 2.1: The relationship between the metrics and number of CRs.

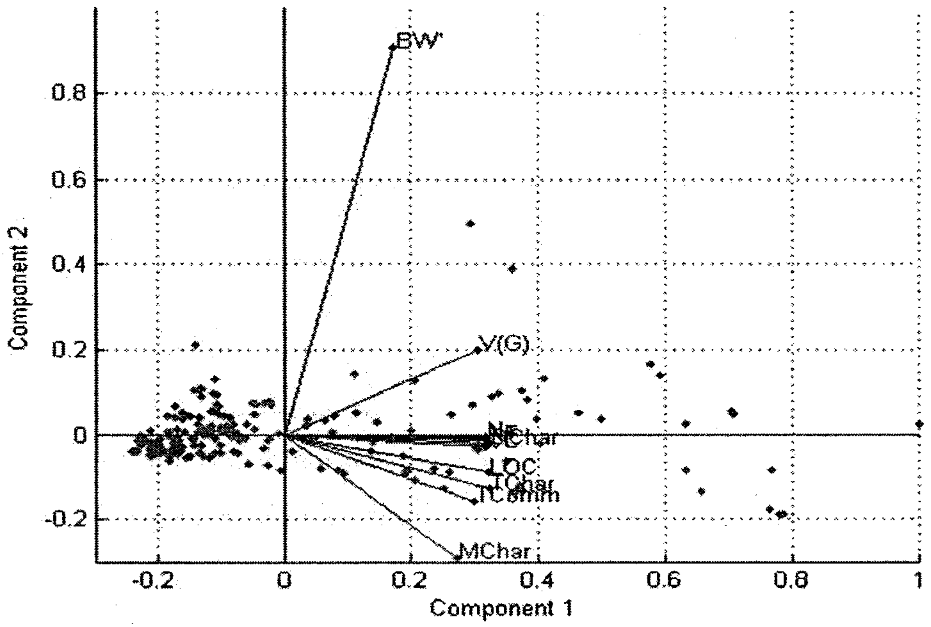

Figure 2.2: Two-dimensional plot of variable coefficients and scores.

Table 2.5: Absolute-value results of resampled paired t test with PCA pretreatment.

	t test (type I error)	t test (type II error)
	vs. LDA	vs. LDA
NDA	1.6656	1.3882
Logistic Regression	0.1304	0.5349
Multiple Linear Regression	1.0977	1.0609
SVM (Polynomial)	1.1543	1.9027
SVM (RBF)	1.1567	0.5309
SVM (Sigmoid)	2.6381	1.4429
	vs. NDA	vs. NDA
Logistic Regression	1.3731	0.5315
Multiple Linear Regression	2.2245	3.4619
SVM (Polynomial)	2.7321	0.0803
SVM (RBF)	3.0483	0.6148
SVM (Sigmoid)	2.9724	0.2211
	vs. Logistic Regression	vs. Logistic Regression
Multiple Linear Regression	1.3211	1.3369
SVM (Polynomial)	1.7492	0.7191
SVM (RBF)	0.0405	0.1845
SVM (Sigmoid)	0.0405	0.1845
	vs. Multiple Linear Regression	vs. Multiple Linear Regression
SVM (Polynomial)	0.6748	2.0658
SVM (RBF)	1.6699	0.0130
SVM (Sigmoid)	2.8554	0.5899
	vs. SVM (Polynomial)	vs. SVM (Polynomial)
SVM (RBF)	0.0452	0.5443
SVM (Sigmoid)	2.3825	0.0881
	vs. SVM (RBF)	vs. SVM (RBF)
SVM (Sigmoid)	1.9211	0.7198

Table 2.6: Absolute-value results of resampled paired *t* test with PCA pretreatment and using the first two significant principal components.

	t test (type I error)	*t* test (type II error)
	vs. LDA	vs. LDA
NDA	1.6656	1.3882
Logistic Regression	1.6560	2.2456
SVM (Polynomial)	3.3642	1.0491
SVM (RBF)	3.0496	1.0105
SVM (Sigmoid)	4.2935	0.4204
	vs. NDA	vs. NDA
Logistic Regression	1.1709	3.1772
SVM (Polynomial)	3.2828	1.2730
SVM (RBF)	3.0664	1.3567
SVM (Sigmoid)	4.0206	0.4859
	vs. Logistic Regression	vs. Logistic Regression
SVM (Polynomial)	3.3619	0.1842
SVM (RBF)	2.6313	0.2801
SVM (Sigmoid)	3.6404	0.7881
	vs. SVM (Polynomial)	vs. SVM (Polynomial)
SVM (RBF)	0.3338	0.0288
SVM (Sigmoid)	0.0525	0.6106
	vs. SVM (RBF)	vs. SVM (RBF)
SVM (Sigmoid)	0.3903	0.7060

Table 2.7: Absolute-value results of resampled paired *t* test with 11 software complexity metrics with and without PCA-pretreatment.

	t test(type I error)	*t* test(type II error)
LDA	0.5167	0.1782
NDA	1.0000	2.6938
Logistic Regression	0.5295	0.2353
Multiple Linear Regression	3.1840	0.6866
SVM (Polynomial)	0.5374	0.3915
SVM (RBF)	0.2134	1.0780
SVM (Sigmoid)	0.2854	0.2130

Table 2.8: Absolute-value results of resampled paired t test with 11 and 2 software complexity metrics.

	t test(type I error)	t test(type II error)
LDA	2.2048	2.4728
NDA	0.1329	2.0252
Logistic Regression	0.2088	0.3632
SVM (Polynomial)	2.7016	0.0303
SVM (RBF)	2.7200	0.4772
SVM (Sigmoid)	1.5541	0.7272

CHAPTER 3

Software Modules Categorization Through Likelihood and Bayesian Analysis of Finite Dirichlet Mixtures

In this chapter we will explore a new unsupervised learning algorithm, finite Dirichlet mixture model, to classify software modules, by employing deterministic maximum likelihood and Bayesian estimation. The selection of the number of clusters for both approaches is based on the Bayesian Information Criterion (BIC). Experimental results are presented using simulated data, as well as the Medical Imaging System (MIS) data sets. A shorter version of this chapter is accepted by IEEE International Conference on Intelligent Systems [90], and extended version is submitted to Journal of Applied Statistics [91].

3.1 Introduction

The increasing availability of data in different fields has triggered the need for its analysis and modeling using statistical approaches [92]. The ultimate goal of these approaches is to describe and explain data with a probabilistic model. Finite mixture models [81,93,94] have been widely used to achieve this goal, since their introduction by Pearson [95], and are now applied in several disciplines. Indeed, finite mixture models allow the clustering of data into groups that are internally homogeneous. Mixture models could also be used to approximate distributions that cannot be modeled by standard parametric families. There are, however, three important issues that need to be addressed when dealing with finite mixture models: the choice of the component's densities, the estimation of the mixture parameters, and the selection of the number of clusters which best describes the data. The component's densities should be chosen depending on the data being examined. For the estimation of the mixture parameters, some researchers have used the method of moments. The common current approaches to solve this problem are, however, based on the maximum likelihood or Bayesian techniques. An important part of the modeling problem is mainly concerned about determining the number of consistent components which best describe the data. For this purpose, many approaches have been suggested. From a computational point of view, these approaches may be divided into two main categories: deterministic and Bayesian methods (See [81] for a detailed survey of selection criteria approaches).

In this thesis, we are interested in modeling data using a rich class of finite mixture distributions called the Dirichlet mixture, which is a multivariate generalization of the Beta mixture. Finite Beta mixtures have been studied by Bouguila *et al.* in [96], highlighting some difficulties when performing the likelihood approach and proposing a Bayesian inference to estimate the parameters. In this thesis, we extend this study to the multidimensional case. Despite the fact that this distribution plays an important role in statistical inference, and also in contrast to the vast amount of theoretical work that exists regarding the characterization of the Dirichlet distribution (for instance, See [97–103]), very little work has been done, however, on its practical applications. The majority of the studies either consider a single Dirichlet distribution [104, 105] or use it as a prior to the multinomial [106–108]. Indeed, many researchers consider finite Gaussian mixtures for data modeling. The Dirichlet mixture, however, could offer better modeling capabilities as shown in [109–111] where it was used as a parent distribution and not as a prior for different image processing tasks.

If the random vector $\boldsymbol{X} = (X_1, \ldots, X_d)$ follows a Dirichlet distribution with parameters $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_d)$, then the joint density function is given by

$$p(\boldsymbol{X}|\boldsymbol{\alpha}) = \frac{\Gamma(|\boldsymbol{\alpha}|)}{\prod_{i=1}^{d} \Gamma(\alpha_i)} \prod_{i=1}^{d} X_i^{\alpha_i - 1} \tag{1}$$

where $\sum_{i=1}^{d} X_i = 1$[1], and $|\boldsymbol{\alpha}| = \sum_{i=1}^{d} \alpha_i, \quad \alpha_i > 0 \quad \forall i = 1 \ldots d$. This distribution is the multivariate extension of the 2-parameter Beta distribution [96]. Unlike the normal distribution, the Dirichlet does not have separate parameters describing the mean and variation. The mean and the variance, however, can be calculated using $\boldsymbol{\alpha}$ as follows

$$\mu_i = E(X_i) = \frac{\alpha_i}{|\boldsymbol{\alpha}|} \tag{2}$$

$$Var(X_i) = \frac{\alpha_i(|\boldsymbol{\alpha}| - \alpha_i)}{|\boldsymbol{\alpha}|^2(|\boldsymbol{\alpha}| + 1)}. \tag{3}$$

[1]The Dirichlet distribution can be extended easily to be defined in any d-dimensional rectangular domain $[a_1, b_1] \times \ldots \times [a_d, b_d]$ where $(a_1, \ldots, a_d) \in \mathbb{R}^d$ and $(b_1, \ldots, b_d) \in \mathbb{R}^d$.

Substituting Eq. (2) into Eq. (1), we may rewrite the Dirichlet distribution as follows

$$p(\boldsymbol{X}||\boldsymbol{\alpha}|, \boldsymbol{\mu}) = \frac{\Gamma(|\boldsymbol{\alpha}|)}{\prod_{i=1}^{d} \Gamma(\mu_i|\boldsymbol{\alpha}|)} \prod_{i=1}^{d} X_i^{\mu_i|\boldsymbol{\alpha}|-1}, \tag{4}$$

where $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_d)$. Note that this alternative parametrization was also adopted in the case of the Beta distribution by Bouguila *et al.* [96], providing interpretable parameters because $\boldsymbol{\mu}$ and $|\boldsymbol{\alpha}|$ represent the mean and a measure of the sharpness of the distribution, respectively [107]. A large value of $|\boldsymbol{\alpha}|$ produces a sharply peaked distribution around the mean $\boldsymbol{\mu}$. And when $|\boldsymbol{\alpha}|$ decreases, the distribution becomes broader as depicted in Fig. 3.1. An additional advantage of this parametrization is that $\boldsymbol{\mu}$ lies within a bounded space, leading to an increase in computational efficiency. Therefore, this parametrization will be adopted throughout the thesis.

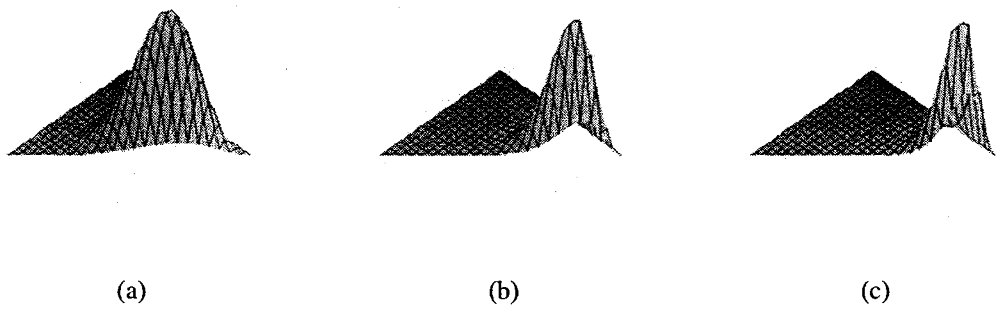

Figure 3.1: The Dirichlet distribution for different parameters. (a) $|\boldsymbol{\alpha}|$=14, $\mu_1 = 0.25$, $\mu_2 = 0.5$, $\mu_3 = 0.25$. (b) $|\boldsymbol{\alpha}|$=21, $\mu_1 = 0.16$, $\mu_2 = 0.68$, $\mu_3 = 0.16$. (c) $|\boldsymbol{\alpha}|$=28, $\mu_1 = 0.125$, $\mu_2 = 0.75$, $\mu_3 = 0.125$.

A Dirichlet mixture with M components is defined as

$$p(\boldsymbol{X}|\xi) = \sum_{j=1}^{M} p(\boldsymbol{X}||\boldsymbol{\alpha}_j|, \boldsymbol{\mu}_j)p(j) \tag{5}$$

where $p(j)$ ($0 < p(j) < 1$ and $\sum_{j=1}^{M} p(j) = 1$) are the mixing parameters and $p(\boldsymbol{X}||\boldsymbol{\alpha}_j|, \boldsymbol{\mu}_j)$ is the Dirichlet distribution. The symbol ξ denotes the entire set of parameters to be estimated, that is

$$\xi = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_M, |\boldsymbol{\alpha}_1|, \ldots, |\boldsymbol{\alpha}_M|, p(1), \ldots, p(M)).$$

This set of parameters can be divided into three subsets $\xi_1 = (|\boldsymbol{\alpha}_1|, \ldots, |\boldsymbol{\alpha}_M|)$, $\xi_2 = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_M)$, and $\xi_3 = (p(1), \ldots, p(M))$. Then, these three different parameters ξ_1, ξ_2 and ξ_3 can be estimated independently.

Most currently used statistical estimation techniques are deterministic. With deterministic approaches, a random sample of observations is drawn from a distribution or a mixture of distributions with unknown parameters assumed to be fixed. In contrast to deterministic methods, Bayesian approaches consider the parameters as random variables and allow probability distributions to be associated with them. In recent years, Bayesian estimation has become feasible due to the development of simulation-based numerical integration techniques such as Markov chain Monte Carlo (MCMC) methods, which simulate required estimates by running appropriate Markov Chains using specific algorithms such as Gibbs sampler.

In this chapter, we consider two procedures for finite Dirichlet mixture estimation, namely, the deterministic maximum likelihood (ML) estimation that will be developed in the next section and the Bayesian estimation that will be explained in details in Section 3.3. Section 3.4 is devoted to an important problem in the case of mixture models, which is the selection of the number of clusters. In Section 3.5 we present our experimental results where both procedures are compared in different applications.

3.2 ML Estimation of a Dirichlet Mixture

Now we consider ML estimation for an M-component mixture of Dirichlet distributions. Given the set of independent vectors $\mathcal{X} = \{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N\}$, the log-likelihood corresponding to an M-component mixture is given by

$$L(\xi, \mathcal{X}) = \log \prod_{i=1}^{N} p(\boldsymbol{X}_i|\xi) = \sum_{i=1}^{N} \log \sum_{j=1}^{M} p(\boldsymbol{X}_i||\boldsymbol{\alpha}_j|, \boldsymbol{\mu}_j) p(j). \tag{6}$$

It is well-known that the ML estimate

$$\hat{\xi}_{ML} = \arg\max_{\xi}\{L(\xi, \mathcal{X})\} \tag{7}$$

cannot be found analytically. The maximization defining the ML estimates is subject to the constraints $0 < p(j) \leq 1$ and $\sum_{j=1}^{M} p(j) = 1$. It is worth pointing out that obtaining ML estimates of the mixture parameters is possible using the expectation-maximization (EM) algorithm and related techniques [112]. The EM algorithm [113, 114] is a general approach to maximum likelihood in the presence of incomplete data. In EM, the "complete" data are considered to be $\boldsymbol{Y}_i = \{\boldsymbol{X}_i, \boldsymbol{Z}_i\}$, where $\boldsymbol{Z}_i = (Z_{i1}, \ldots, Z_{iM})$ with

$$Z_{ij} = \begin{cases} 1 & \text{if } \boldsymbol{X}_i \text{ belongs to class } j \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

constituting the "missing" data. The relevant assumption is that the density of an observation $\boldsymbol{X}_i$ given Z_i is given by $\prod_{j=1}^{M} p(\boldsymbol{X}_i||\boldsymbol{\alpha}_j|, \boldsymbol{\mu}_j)^{Z_{ij}}$.

The resulting *complete-data log-likelihood* is therefore

$$L(\boldsymbol{\xi}, \mathcal{Z}, \mathcal{X}) = \sum_{i=1}^{N} \sum_{j=1}^{M} Z_{ij} \log(p(\boldsymbol{X}_i||\boldsymbol{\alpha}_j|, \boldsymbol{\mu}_j)p(j)), \tag{9}$$

where $\mathcal{Z} = \{Z_1, \ldots, Z_N\}$.

The EM algorithm produces a sequence of estimates $\{\boldsymbol{\xi}^t, t = 0, 1, 2 \ldots\}$ by applying two steps in alternation (until some convergence criterion is satisfied)

1. **E-step:** Compute $\hat{Z}_{ij}$ given the parameter estimates from the initialization

$$\hat{Z}_{ij} = \frac{p(\boldsymbol{X}_i||\boldsymbol{\alpha}_j|, \boldsymbol{\mu}_j)p(j)}{\sum_{j=1}^{M} p(\vec{X}_i||\boldsymbol{\alpha}_j|, \boldsymbol{\mu}_j)p(j)} \tag{10}$$

2. **M-step:** Update the parameter estimates according to

$$\hat{\boldsymbol{\xi}} = \arg\max_{\boldsymbol{\xi}} L(\boldsymbol{\xi}, \mathcal{Z}, \mathcal{X}) \tag{11}$$

The quantity $\hat{Z}_{ij}$ is the conditional expectation of Z_{ij} given the observation $\boldsymbol{X}_i$ and the parameter vector $\boldsymbol{\xi}$. The value Z_{ij}^* of $\hat{Z}_{ij}$ at a maximum of Eq. (9) is the conditional probability that observation i belongs to class j (the *posterior* probability); the classification of an observation X_i is taken to be $\{k/Z_{ik}^* = \max_j Z_{ij}^*\}$, which is the Bayes rule. When we maximize Eq. (11) by taking into consideration the constraints $\sum_{j=1}^{M} p(j) = 1$ and $p(j) > 0 \quad \forall j \in [1, M]$, we obtain

$$p(j)^{(t)} = \frac{1}{N} \sum_{i=1}^{N} \hat{Z}_{ij}^{(t-1)} \tag{12}$$

3.2.1 Estimation of the $\boldsymbol{\xi}_1$ Parameters

For fixed $\boldsymbol{\mu}_j$, the likelihood for $\boldsymbol{\xi}_1$ alone is given by

$$p(\mathcal{X}|\boldsymbol{\xi}_1) \propto \prod_{i=1}^{N} \left[\sum_{j=1}^{M} p(j) p(\boldsymbol{X}_i||\boldsymbol{\alpha}_j|) \right] \tag{13}$$

where

$$p(\boldsymbol{X}_i||\boldsymbol{\alpha}_j|) = \frac{\Gamma(|\boldsymbol{\alpha}_j|)}{\prod_{l=1}^{d} \Gamma(\mu_{jl}|\boldsymbol{\alpha}_j|)} \prod_{l=1}^{d} X_{il}^{\mu_{jl}|\boldsymbol{\alpha}_j|-1} \tag{14}$$

Maximizing the log-likelihood function is equivalent to solving the following equation

$$\frac{\partial \log p(\mathcal{X}|\xi_1)}{\partial |\alpha_j|} = 0, \qquad j = 1, \dots, M \tag{15}$$

The first-order partial derivative of $\log p(\mathcal{X}|\xi_1)$ with respect to $|\alpha_j|$ is given by

$$\begin{aligned}
\frac{\partial \log p(\mathcal{X}|\xi_1)}{\partial |\alpha_j|} &= \sum_{i=1}^{N} \frac{\partial}{\partial |\alpha_j|} \log \left[\sum_{j=1}^{M} p(j) p(\boldsymbol{X}_i \| \alpha_j|) \right] = \sum_{i=1}^{N} \frac{\frac{\partial}{\partial |\alpha_j|} \left[\sum_{j=1}^{M} p(j) p(\boldsymbol{X}_i \| \alpha_j|) \right]}{\sum_{j=1}^{M} p(j) p(\boldsymbol{X}_i \| \alpha_j|)} \\
&= \sum_{i=1}^{N} \hat{Z}_{ij} \frac{\frac{\partial}{\partial |\alpha_j|} \left[p(\boldsymbol{X}_i \| \alpha_j|) \right]}{p(\boldsymbol{X}_i \| \alpha_j|)} = \sum_{i=1}^{N} \hat{Z}_{ij} \frac{\partial}{\partial |\alpha_j|} \log p(\boldsymbol{X}_i \| \alpha_j|) \Big] \\
&= \left(\Psi(|\alpha_j|) - \sum_{l=1}^{d} \mu_{jl} \Psi(\mu_{jl} |\alpha_j|) \right) \sum_{i=1}^{N} \hat{Z}_{ij} + \sum_{i=1}^{N} \left(\hat{Z}_{ij} \sum_{l=1}^{d} \mu_{jl} \log(X_{il}) \right),
\end{aligned}$$

where Ψ denotes the digamma function. Note that from Eq. (16), it is clear that we do not have a closed-form solution to Eq. (15). Thus, to estimate $|\alpha_j|$ we use the Newton-Raphson method as follows

$$|\alpha_j|^{(t)} = |\alpha_j|^{(t-1)} - \left(\frac{\partial^2 \log p(\mathcal{X}|\xi_1^{(t-1)})}{\partial^2 |\alpha_j|} \right)^{-1} \frac{\partial \log p(\mathcal{X}|\xi_1^{(t-1)})}{\partial |\alpha_j|}. \tag{16}$$

On the other hand, the second-order partial derivative of $\log p(\mathcal{X}|\xi_1)$ with respect to $|\alpha_j|$ is given by

$$\frac{\partial^2 \log p(\mathcal{X}|\xi_1)}{\partial^2 |\alpha_j|} = \left(\Psi'(|\alpha_j|) - \sum_{l=1}^{d} \mu_{jl}^2 \Psi'(\mu_{jl} |\alpha_j|) \right) \sum_{i=1}^{N} \hat{Z}_{ij} \tag{17}$$

3.2.2 Estimation of the ξ_2 Parameters

For fixed $|\alpha_j|$, the likelihood function for ξ_2 alone is given by

$$p(\mathcal{X}|\xi_2) = \prod_{i=1}^{N} \left[\sum_{j=1}^{M} p(j) p(\boldsymbol{X}_i | \boldsymbol{\mu}_j) \right], \tag{18}$$

where

$$p(\boldsymbol{X}_i | \boldsymbol{\mu}_j) = \frac{\Gamma(|\alpha_j|)}{\prod_{l=1}^{d} \Gamma(\mu_{jl} |\alpha_j|)} \prod_{l=1}^{d} X_{il}^{\mu_{jl}|\alpha_j| - 1} \propto \prod_{l=1}^{d} \frac{X_{il}^{\mu_{jl}|\alpha_j| - 1}}{\Gamma(\mu_{jl} |\alpha_j|)}. \tag{19}$$

The maximization of the likelihood function $\log p(\mathcal{X}|\xi_2)$, subject to the constraints $\sum_{l=1}^{d} \mu_{jl} = 1,\ j \in \{1, \dots, M\}$, leads to the maximization of the following objective function

$$\Phi_{Mean}(\mathcal{X}, \xi_2, \Lambda) = \log p(\mathcal{X}|\xi_2) + \Lambda_1 (1 - \sum_{l=1}^{d} \mu_{1l}) + \ldots + \Lambda_M (1 - \sum_{l=1}^{d} \mu_{Ml}), \tag{20}$$

where $\boldsymbol{\Lambda} = (\Lambda_1, \Lambda_2, \dots, \Lambda_M)$ denotes the Lagrange multipliers vector.

To solve this optimization problem, we first need to determine the solutions to the following equations

$$\frac{\partial}{\partial \mu_{jl}} \Phi_{Mean}(\mathcal{X}, \boldsymbol{\xi}_2, \boldsymbol{\Lambda}) = 0 \tag{21}$$

$$\frac{\partial}{\partial \Lambda} \Phi_{Mean}(\mathcal{X}, \boldsymbol{\xi}_2, \boldsymbol{\Lambda}) = 0 \tag{22}$$

where $j = 1, \dots, M$ and $l = 1, \dots, d$.

Setting the equation

$$\begin{aligned} \frac{\partial}{\partial \mu_{jl}} \Phi_{Mean}(\mathcal{X}, \boldsymbol{\xi}_2, \boldsymbol{\Lambda}) &= \sum_{i=1}^{N} \hat{Z}_{ij} \frac{\partial}{\partial \mu_{jl}} \log(p(\boldsymbol{X}_i | \boldsymbol{\mu}_j)) - \Lambda_j \\ &= |\boldsymbol{\alpha}_j| \sum_{i=1}^{N} \hat{Z}_{ij} \Big(\log(X_{il}) - \Psi(\mu_{jl} |\boldsymbol{\alpha}_j|) \Big) - \Lambda_j \end{aligned} \tag{23}$$

to 0 yields

$$\mu_{jl}^{(t)} = \frac{\mu_{jl}^{(t-1)} |\boldsymbol{\alpha}_j|^{(t)} \sum_{i=1}^{N} \hat{Z}_{ij}^{(t)} \Big(log(X_{il}) - \Psi(\mu_{jl}^{(t-1)} |\boldsymbol{\alpha}_j|^{(t)}) \Big)}{\Lambda_j}. \tag{24}$$

Similarly, setting the equation

$$\frac{\partial}{\partial \Lambda_j} \Phi_{Mean}(\mathcal{X}, \boldsymbol{\xi}_2, \Lambda) = 1 - \sum_{l=1}^{d} \mu_{jl} \tag{25}$$

to 0 yields

$$\sum_{l=1}^{d} \mu_{jl} = 1. \tag{26}$$

Substituting Eq. (24) into Eq. (26), we obtain

$$\frac{\sum_{l=1}^{d} \mu_{jl} |\boldsymbol{\alpha}_j| \sum_{i=1}^{N} \hat{Z}_{ij} \Big(\log(X_{il}) - \Psi(\mu_{jl} |\boldsymbol{\alpha}_j|) \Big)}{\Lambda_j} = 1. \tag{27}$$

Therefore, we get

$$\Lambda_j = \sum_{l=1}^{d} \mu_{jl} |\boldsymbol{\alpha}_j| \sum_{i=1}^{N} \hat{Z}_{ij} \Big(\log(X_{il}) - \Psi(\mu_{jl} |\boldsymbol{\alpha}_j|) \Big). \tag{28}$$

And substituting Eq. (28) in Eq. (24), we obtain

$$\mu_{jl}^{(t)} = \frac{\mu_{jl}^{(t-1)} \sum_{i=1}^{N} \hat{Z}_{ij}^{(t)} \Big(\log(X_{il}) - \Psi(\mu_{jl}^{(t-1)} |\boldsymbol{\alpha}_j|^{(t)}) \Big)}{\sum_{l=1}^{d} \Big[\mu_{jl}^{(t-1)} \sum_{i=1}^{N} \hat{Z}_{ij}^{(t)} \Big(\log(X_{il}) - \Psi(\mu_{jl}^{(t-1)} |\boldsymbol{\alpha}_j|^{(t)}) \Big) \Big]}. \tag{29}$$

Having all these estimation equations, the complete algorithm for the parameters estimation may be summarized as follows:

1. Apply the initialization algorithm proposed in [110]

2. **E-Step**: Compute the *a posteriori* probabilities:

$$\hat{Z}_{ij} = \frac{p(\boldsymbol{X}_i|\boldsymbol{\xi}_j)p(j)}{\sum_{l=1}^{M} p(\boldsymbol{X}_i|\boldsymbol{\xi}_l)p(l)}$$

3. **M-Step**:

   (a) Update the $|\alpha_j|$ using Eq. (16), $j = 1, \dots, M$.

   (b) Update the μ_{jl} using Eq. (29), $j = 1, \dots, M$, and $l = 1, \dots, d$.

   (c) Update the $p(j)$ using Eq. (12), $j = 1, \dots, M$.

4. If the convergence test is passed, terminate, else go to 2.

3.3 Bayesian Estimation of a Dirichlet Mixture

Bayesian estimation is based on learning from data using Bayes's theorem in order to combine both the prior information and the information brought by the data to produce the posterior distribution [115, 116]. The prior information represents our prior belief about the parameters before looking at the data. The posterior distribution summarizes our belief about the parameters after we have analyzed the data. The posterior distribution can be expressed as

$$p(\boldsymbol{\xi}|\mathcal{X}) \propto p(\mathcal{X}|\boldsymbol{\xi})p(\boldsymbol{\xi}). \tag{30}$$

From Eq. (30), we can see that Bayesian estimation requires a prior distribution $p(\boldsymbol{\xi})$ specification for the mixture parameters. As in the case of the EM algorithm, the introduction of the $\boldsymbol{Z}$ vectors simplifies the Bayesian analysis [117]. This is done by associating with each observation $\boldsymbol{X}_i$ a missing multinomial variable $\boldsymbol{Z}_i \sim \mathcal{M}(1; \hat{Z}_{i1}, \dots, \hat{Z}_{iM})$, and the complete MCMC algorithm is given by [118]

1. Initialization

2. Step t: For t=1,...

   (a) Generate $\boldsymbol{Z}_i^{(t)} \sim \mathcal{M}(1; \hat{Z}_{i1}^{(t-1)}, \dots, \hat{Z}_{iM}^{(t-1)})$

   (b) Generate $\boldsymbol{\xi}_3^{(t)}$ from $p(\boldsymbol{\xi}_3|\mathcal{Z}^{(t)})$

   (c) Generate $(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2)^{(t)}$ from $p(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2|\mathcal{Z}^{(t)}, \mathcal{X})$

First we start with the distribution $p(\boldsymbol{\xi}_3|\mathcal{Z})$, which is given by

$$p(\boldsymbol{\xi}_3|\mathcal{Z}) \quad \propto \quad p(\boldsymbol{\xi}_3)p(\mathcal{Z}|\boldsymbol{\xi}_3) \tag{31}$$

and then we determine $p(\boldsymbol{\xi}_3)$ and $p(\mathcal{Z}|\boldsymbol{\xi}_3)$. It is known that the vector $\boldsymbol{\xi}_3$ is defined on the simplex $\{(p(1),\dots,p(M)) : \sum_{j=1}^{M-1} p(j) < 1\}$, thus a natural choice, as a prior, for this vector would be the Dirichlet distribution [118]

$$p(\boldsymbol{\xi}_3) = \frac{\Gamma(\sum_{j=1}^{M}\eta_j)}{\prod_{j=1}^{M}\Gamma(\eta_j)}\prod_{j=1}^{M} p(j)^{\eta_j-1} \tag{32}$$

where $\boldsymbol{\eta} = (\eta_1,\dots,\eta_M)$ is the parameter vector of the Dirichlet distribution. Moreover, we have

$$p(\mathcal{Z}|\boldsymbol{\xi}_3) \quad = \quad \prod_{i=1}^{N} p(\boldsymbol{Z}_i|\boldsymbol{\xi}_3) = \prod_{i=1}^{N} p(1)^{Z_{i1}}\dots p(M)^{Z_{iM}} = \prod_{i=1}^{N}\prod_{j=1}^{M} p(j)^{Z_{ij}} = \prod_{j=1}^{M} p(j)^{n_j}.$$

Hence

$$\begin{aligned} p(\boldsymbol{\xi}_3|\mathcal{Z}) \quad &= \quad \frac{\Gamma(\sum_{j=1}^{M}\eta_j)}{\prod_{j=1}^{M}\Gamma(\eta_j)}\prod_{j=1}^{M} p(j)^{\eta_j-1}\prod_{j=1}^{M} p(j)^{n_j} = \frac{\Gamma(\sum_{j=1}^{M}\eta_j)}{\prod_{j=1}^{M}\Gamma(\eta_j)}\prod_{j=1}^{M} p(j)^{\eta_j+n_j-1} \\ &\propto \quad \mathcal{D}(\eta_1+n_1,\dots,\eta_M+n_M) \end{aligned} \tag{33}$$

where $\mathcal{D}$ is a Dirichlet distribution with parameters $(\eta_1+n_1,\dots,\eta_M+n_M)$. Note that both the prior and the posterior distributions, $p(\boldsymbol{\xi}_3)$ and $p(\boldsymbol{\xi}_3|\mathcal{Z})$ are Dirichlet. In this case we say that the Dirichlet distribution is a conjugate prior for the mixture proportions. We held the hyperparameters η_j fixed at 1 which is a classical and a reasonable choice.

For a mixture of Dirichlet distributions, it is therefore possible to associate with each $|\boldsymbol{\alpha}_j|$ a prior $p_j(|\boldsymbol{\alpha}_j|)$ and with each $\boldsymbol{\mu}_j$ a prior $p_j(\boldsymbol{\mu}_j)$. For the same reasons as the mixing proportion, we can select a Dirichlet prior for $\boldsymbol{\mu}_j$

$$p(\boldsymbol{\mu}_j) = \frac{\Gamma(\sum_{l=1}^{d}\vartheta_l)}{\prod_{l=1}^{M}\Gamma(\vartheta_l)}\prod_{l=1}^{d}\mu_{jl}^{\vartheta_l-1}. \tag{34}$$

For $|\boldsymbol{\alpha}_j|$, we adopt a vague prior of inverse Gamma shape $p(|\boldsymbol{\alpha}_j|^{-1}) \sim \mathcal{G}(1,1)$ as proposed in [119]

$$p(|\boldsymbol{\alpha}_j|) \propto |\boldsymbol{\alpha}_j|^{-3/2}\exp\big(-1/(2|\boldsymbol{\alpha}_j|)\big). \tag{35}$$

Having these priors, the posterior distribution is then given by

$$\begin{aligned} p(|\boldsymbol{\alpha}_j|,\boldsymbol{\mu}_j|\mathcal{Z},\mathcal{X}) \quad &\propto \quad p(|\boldsymbol{\alpha}_j|)p(\boldsymbol{\mu}_j)\prod_{Z_{ij}=1} p(\boldsymbol{X}_i||\boldsymbol{\alpha}_j|,\boldsymbol{\mu}_j) \\ &\propto \quad |\boldsymbol{\alpha}_j|^{-3/2}\exp\big(-1/(2|\boldsymbol{\alpha}_j|)\big)\frac{\Gamma(\sum_{l=1}^{d}\vartheta_l)}{\prod_{l=1}^{d}\Gamma(\vartheta_l)}\prod_{l=1}^{d}\mu_{jl}^{\vartheta_l-1} \\ &\times \quad \left(\frac{\Gamma(|\boldsymbol{\alpha}_j|)}{\prod_{l=1}^{d}\Gamma(\mu_{jl}|\boldsymbol{\alpha}_j|)}\right)^{n_j}\prod_{l=1}^{d}\left(\prod_{Z_{ij}=1} X_{il}\right)^{\mu_{jl}|\boldsymbol{\alpha}_j|-1} \end{aligned} \tag{36}$$

The hyperparameters ϑ_l are chosen to be equal to 1. Having all these posterior probabilities in hand, the steps of the Gibbs sampler are as follows

1. Initialization
2. Step t: For t=1,...
   (a) Generate $Z_i^{(t)} \sim \mathcal{M}(1; \hat{Z}_{i1}^{(t-1)}, \ldots, \hat{Z}_{iM}^{(t-1)})$
   (b) Compute $n_j^{(t)} = \sum_{i=1}^{N} \mathbb{I}_{Z_{ij}^{(t)}=j}$
   (c) Generate $P^{(t)}$ from Eq. (33)
   (d) Generate $(|\boldsymbol{\alpha}_j|, \boldsymbol{\mu}_j)^{(t)}$ $(j = 1, \ldots, M)$ from Eq. 36 using the Metropolis-Hastings (M-H) algorithm.

The M-H algorithm, originated in the 1950's in the literature of statistical physics [120–122], offers a solution to the problem of simulating from the posterior distribution. Starting from point $(|\boldsymbol{\alpha}_j|^{(0)}, \boldsymbol{\mu}_j^{(0)})$, the corresponding Markov chain explores the surface of the posterior distribution. At iteration t, the steps of the M-H algorithm can be described as follows

1. Generate $(|\tilde{\boldsymbol{\alpha}}_j|, \tilde{\boldsymbol{\mu}}_j) \sim q(|\boldsymbol{\alpha}_j|, \boldsymbol{\mu}_j||\boldsymbol{\alpha}_j|^{(t-1)}, \boldsymbol{\mu}_j^{(t-1)})$ and $U \sim \mathcal{U}_{[0,1]}$
2. Compute $r = \dfrac{p\left(|\tilde{\boldsymbol{\alpha}}_j|,\tilde{\boldsymbol{\mu}}_j|\mathcal{Z},\mathcal{X}\right) q\left(|\boldsymbol{\alpha}_j|^{(t-1)},\boldsymbol{\mu}_j^{(t-1)}||\tilde{\boldsymbol{\alpha}}_j|,\tilde{\boldsymbol{\mu}}_j\right)}{p\left(|\boldsymbol{\alpha}_j|^{(t-1)},\boldsymbol{\mu}_j^{(t-1)}|\mathcal{Z},\mathcal{X}\right) q\left(|\tilde{\boldsymbol{\alpha}}_j|,\tilde{\boldsymbol{\mu}}_j||\boldsymbol{\alpha}_j|^{(t-1)},\boldsymbol{\mu}_j^{(t-1)}\right)}$
3. If $r < u$ then $(|\boldsymbol{\alpha}_j|^{(t)}, \boldsymbol{\mu}_j^{(t)}) = (|\tilde{\boldsymbol{\alpha}}_j|, \tilde{\boldsymbol{\mu}}_j)$ else $(|\boldsymbol{\alpha}_j|^{(t)}, \boldsymbol{\mu}_j^{(t)}) = (|\boldsymbol{\alpha}_j|^{(t-1)}, \boldsymbol{\mu}_j^{(t-1)})$

A major problem with this algorithm is the need to choose the proposal distribution q. The most generic proposal is the random walk Metropolis-Hastings algorithm, where each unconstrained parameter is the mean of the proposal distribution for the new value. Since $|\tilde{\boldsymbol{\alpha}}_j| > 0$, we have chosen the following proposal

$$|\tilde{\boldsymbol{\alpha}}_j| \sim \mathcal{LN}(\log(|\boldsymbol{\alpha}_j|^{(t-1)}), \sigma^2), \tag{37}$$

where $\mathcal{LN}(\log(|\boldsymbol{\alpha}_j|^{(t-1)}), \sigma^2)$ denotes the log-normal distribution with mean $\log(|\boldsymbol{\alpha}_j|^{(t-1)})$, and variance σ^2 chosen to be equal to 0.01. Note that Eq. (37) is equivalent to

$$\log(|\tilde{\boldsymbol{\alpha}}_j|) = \log(|\boldsymbol{\alpha}_j|^{(t-1)}) + \epsilon_j, \tag{38}$$

where $\epsilon_j \sim \mathcal{N}(0, \sigma^2)$. However, for constrained parameters, this proposal is not efficient [123]. This is the case for the parameter μ_{jl} (since μ_{jl} belongs to the simplex $[0, 1]$). To circumvent this problem, we first transform $\mu_{jl}, l = 1, \ldots, d+1, j = 1, \ldots, M$ to $\mu^*_{jl} = \log(\mu_{jl}/(1-\mu_{jl}))$, and then we use the following proposal

$$\tilde{\boldsymbol{\mu}^*_j} \sim \mathcal{LN}(\log(\boldsymbol{\mu}^{*(t-1)}_j), \Sigma^2), \tag{39}$$

where $\Sigma^2 = diag[0.01, \ldots, 0.01]$ is a diagonal matrix.

With these proposals, the random walk M-H algorithm is composed of the following steps:

1. Generate $|\tilde{\boldsymbol{\alpha}_j}| \sim \mathcal{LN}(\log(|\boldsymbol{\alpha}_j|^{(t-1)}), \sigma^2)$, $\tilde{\mu^*_{jl}} \sim \mathcal{LN}(\log({\mu^*_{jl}}^{(t-1)}), \sigma^2)$ and $U \sim \mathcal{U}_{[0,1]}$.
2. Compute $r = \dfrac{p\Big(|\tilde{\boldsymbol{\alpha}_j}|, \tilde{\boldsymbol{\mu}^*_j}|\mathcal{Z},\mathcal{X}\Big)\mathcal{LN}(\boldsymbol{\mu}^{*(t-1)}_j|\log(\tilde{\boldsymbol{\mu}^*_j}),\Sigma^2)\mathcal{LN}(|\boldsymbol{\alpha}_j|^{(t-1)}|\log(|\tilde{\boldsymbol{\alpha}_j}|),\sigma^2)}{p\Big(|\tilde{\boldsymbol{\alpha}_j}|^{(t-1)}, \tilde{\boldsymbol{\mu}^*_j}^{(t-1)}|\mathcal{Z},\mathcal{X}\Big)\mathcal{LN}(\tilde{\boldsymbol{\mu}^*_j}|\log(\boldsymbol{\mu}^{*(t-1)}_j),\Sigma^2)\mathcal{LN}(|\tilde{\boldsymbol{\alpha}_j}||\log(|\boldsymbol{\alpha}_j|^{(t-1)}),\sigma^2)}$
3. If $r < u$ then $(|\boldsymbol{\alpha}_j|^{(t)}, \boldsymbol{\mu}^{*(t)}_j) = (|\tilde{\boldsymbol{\alpha}_j}|, \tilde{\boldsymbol{\mu}^*_j})$ else $(|\boldsymbol{\alpha}_j|^{(t)}, \boldsymbol{\mu}^{*(t)}_j) = (|\boldsymbol{\alpha}_j|^{(t-1)}, \boldsymbol{\mu}^{*(t-1)}_j)$

where

$$\begin{aligned} p(|\boldsymbol{\alpha}_j|, \boldsymbol{\mu}^*_j|\mathcal{Z},\mathcal{X}) &\propto |\boldsymbol{\alpha}_j|^{-3/2}\exp\big(-1/(2|\boldsymbol{\alpha}_j|)\big)\frac{\Gamma(\sum_{l=1}^d \vartheta_l)}{\prod_{l=1}^d \Gamma(\vartheta_l)}\prod_{l=1}^d T(\mu^*_{jl})^{\vartheta_l - 1} \\ &\times \left(\frac{\Gamma(|\boldsymbol{\alpha}_j|)}{\prod_{l=1}^d \Gamma(T(\mu^*_{jl})|\boldsymbol{\alpha}_j|)}\right)^{n_j}\prod_{l=1}^d\Big(\prod_{Z_{ij}=1} X_{il}\Big)^{T(\mu^*_{jl})|\boldsymbol{\alpha}_j|-1}\prod_{l=1}^d J(\mu^*_{jl}) \end{aligned}$$

with $T(\mu^*_{jl}) = e^{\mu^*_{jl}}/(1+e^{\mu^*_{jl}})$, and $J(\mu^*_{jl}) = e^{\mu^*_{jl}}/(1+e^{\mu^*_{jl}})^2$ denotes the Jacobian of the transformation $T(\mu^*_{jl})$.

Having our algorithm in hand, an important problem that needs to be addressed now is to determine the number of iterations needed to reach convergence [124]. This problem is discussed next.

3.3.1 Convergence

The development of appropriate diagnostic tools to establish Bayesian estimation convergence is an active research area, and different qualitative and quantitative approaches have been proposed in the literature [125, Chapter 6] [126–129]. Note that the convergence in the Bayesian case refers to the convergence to a density and not to a single point as in the case of deterministic approaches. In general, it is not possible to provide a reliable convergence diagnostic through a single sequence of simulation [128]. Indeed, it is better to use two or more parallel chains to allow a complete coverage of the parameters space [130]. One of the most successfully used approaches was presented in [128]. However, this approach is useful only for univariate problems. Fortunately, an extension to multivariate problems was proposed in [129], and was applied with success in the case of finite multivariate t-mixtures [131]. The method of Brooks and Gelman [129] requires the simulation of m parallel chains with over-dispersed initial values, and then the comparison of between and within variances of these chains. Therefore, we define

$$\hat{V} = \frac{L-1}{L}W + \left(1 + \frac{1}{m}\right) B/L \tag{40}$$

as the posterior variance-covariance matrix of $\boldsymbol{\xi}$, where the length of each simulated sequence is $2L$ and

$$W = \frac{1}{m(L-1)} \sum_{j=1}^{m} \sum_{l=L+1}^{2L} (\boldsymbol{\xi}_j^{(t)} - \bar{\boldsymbol{\xi}}_{j\cdot})(\boldsymbol{\xi}_j^{(t)} - \bar{\boldsymbol{\xi}}_{j\cdot})^T \tag{41}$$

$$B/L = \frac{1}{m-1} \sum_{j=1}^{m} (\bar{\boldsymbol{\xi}}_{j\cdot} - \bar{\boldsymbol{\xi}}_{\cdot\cdot})(\bar{\boldsymbol{\xi}}_{j\cdot} - \bar{\boldsymbol{\xi}}_{\cdot\cdot})^T \tag{42}$$

$$\bar{\boldsymbol{\xi}}_{j\cdot} = \frac{1}{L} \sum_{l=L+1}^{2L} \boldsymbol{\xi}_j^{(t)} \tag{43}$$

$$\bar{\boldsymbol{\xi}}_{\cdot\cdot} = \frac{1}{m} \sum_{j=1}^{m} \bar{\boldsymbol{\xi}}_{j\cdot} \tag{44}$$

Then the value of the following measure, called multiple potential scale reduction factor (MPSRF) [129], is used to test the convergence

$$\hat{R} = \frac{L-1}{L} + \left(1 + \frac{1}{n}\right) \lambda_1, \tag{45}$$

where λ_1 is the largest eigenvalue of $W^{-1}B/L$. Note that as the simulation converge $\hat{R}$ will tend to 1 (See [129] for more details about this convergence diagnostic approach).

3.4 Selection of the Number of Clusters

The choice of the number of components M affects the flexibility of the model. For the selection of the number of clusters, we use integrated (or marginal) likelihood that has been used by both deterministic and Bayesian approaches [59, 132–135], and it is defined as

$$p(\mathcal{X}|M) = \int \pi(\boldsymbol{\xi}|\mathcal{X}, M) d\boldsymbol{\xi} = \int p(\mathcal{X}|\boldsymbol{\xi}, M) \pi(\boldsymbol{\xi}|M) d\boldsymbol{\xi}, \quad . \tag{46}$$

where $\boldsymbol{\xi}$ is the vector of parameters of a finite mixture model, $\pi(\boldsymbol{\xi}|M)$ is its prior density, and $p(\mathcal{X}|\boldsymbol{\xi}, M)$ is the likelihood function.

The main problem now is reduced to computing the integrated likelihood. In order to solve this problem, let $\hat{\boldsymbol{\xi}}$ denote the posterior mode satisfying

$$\nabla \log(\pi(\hat{\boldsymbol{\xi}}|\mathcal{X}, M)) = 0 \tag{47}$$

where $\nabla \log(\pi(\hat{\xi}|\mathcal{X}, M))$ denotes the gradient of $\log(\pi(\hat{\xi}|\mathcal{X}, M))$ evaluated at $\xi = \hat{\xi}$. The Hessian matrix of $-\log(\pi(\hat{\xi}|\mathcal{X}, M))$ evaluated at $\xi = \hat{\xi}$ is denoted by $H(\hat{\xi})$. To approximate the integral given by Eq. (46), the integrand is expanded in a second-order Taylor series about the point $\xi = \hat{\xi}$, and the Laplace approximation gives

$$\begin{aligned} p(\mathcal{X}|M) &= \int \exp\Big(\log(\pi(\hat{\xi}|\mathcal{X}, M) - \frac{1}{2}(\Theta - \hat{\xi})^T H(\hat{\xi})(\xi - \hat{\xi})\Big) d\xi \\ &= \pi(\hat{\xi}|\mathcal{X}, M) \int \exp\Big(-\frac{1}{2}(\xi - \hat{\xi})^T H(\hat{\xi})(\xi - \hat{\xi})\Big) d\xi \\ &= \pi(\hat{\xi}|\mathcal{X}, M)(2\pi)^{\frac{N_p}{2}} \sqrt{|H(\hat{\xi})|} \\ &= p(\mathcal{X}|\hat{\xi}, M)\pi(\hat{\xi}|M)(2\pi)^{\frac{N_p}{2}} \sqrt{|H(\hat{\xi})|}, \end{aligned} \tag{48}$$

where N_p is the number of parameters to be estimated, which is equal to $(d+2)M$ in our case, and $|H(\hat{\xi})|$ denotes the determinant of the Hessian matrix. Note that the Laplace approximation is very accurate as shown by Kass *et al.* [133, 136]. Indeed, the relative error of this approximation, given by

$$\frac{p(\mathcal{X}|M)_{Laplace} - p(\mathcal{X}|M)_{correct}}{p(\mathcal{X}|M)_{correct}} \tag{49}$$

is $O_p(1/N)$. For numerical reasons, it is better to work with the Laplace approximation on the logarithm scale. Taking logarithms, we can rewrite Eq. (48) as

$$\log(p(\mathcal{X}|M)) = \log(p(\mathcal{X}|\hat{\xi}, M)) + \log(\pi(\hat{\xi}|M)) + \frac{N_p}{2}\log(2\pi) + \frac{1}{2}\log(|H(\hat{\xi})|). \tag{50}$$

In order to compute the Laplace approximation, we have to determine $\hat{\xi}$ and $H(\hat{\xi})$. However, in many practical situations an analytic solution is not available. Furthermore, the computation of $|H(\hat{\xi})|$ is difficult especially for high-dimensional data. Therefore, we use another efficient approximation [133] which can be deduced from Eq. (50) by retaining only the terms that increase linearly with N [134], and we obtain

$$\log(p(\mathcal{X}|M)) = \log(p(\mathcal{X}|\hat{\xi}, M)) - \frac{N_p}{2}\log(N), \tag{51}$$

which is the Bayesian information criterion (BIC) proposed by Schwarz [137]. This criterion coincides formally (but not conceptually) with the minimum description length criterion (MDL) proposed by Rissanen [86]. Using Eq. (51), the number of components in the mixture model is given by

$$\{M / \log(p(\mathcal{X}|M)) = \max_M \log(p(\mathcal{X}|M)), M = M_{min}, \ldots, M_{max}\}.$$

3.5 Experimental Results

3.5.1 Synthetic Data

We first use synthetic data to examine and compare the properties and effectiveness of both the likelihood and the Bayesian approaches for finite Dirichlet mixture models estimation and selection. Four synthetic two-dimensional data sets were generated using different parameters. The parameters of the four generated data sets are listed in Table 3.1, and the resulting mixtures are shown in Fig. 3.2.

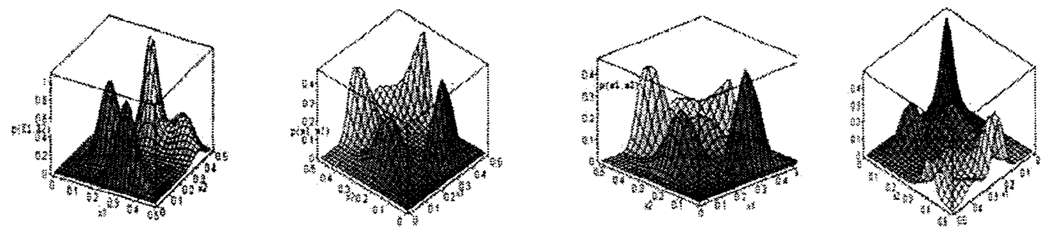
(a) (b) (c) (d)

Figure 3.2: Mixture densities for the generated data sets

Then, we estimate the parameters and the number of clusters of the mixtures representing these data sets. Table 3.2 gives the estimated parameters using both the likelihood and Bayesian approaches. It is worth noting that the estimates obtained by both approaches are similar and very accurate. Fig. 3.3-3.4 depict the computed number of clusters for the generated data sets using the likelihood and Bayesian approaches, respectively. Note that in Fig. 3.3 the values Further, we run 20 independent parallel chains of 7000 iterations each to monitor the convergence of the proposed Bayesian algorithm. The multiple potential scale reduction factor discussed in subsection 3.3.1 came down to 1 within 3250, 3600, 3820 and 4200 iterations for data sets 1, 2, 3 and 4, respectively. It is clear that the correct number of clusters is favored for all data sets, yet the Bayesian approach appears to be better at discriminating between models.

3.5.2 Real Application

Nowadays software products play an important role in different disciplines such as medicine and telecommunications. Improving the quality and reliability of these software products is an active research area, and different approaches, generally based on data analysis techniques, have been proposed [138, 139].

Table 3.1: Parameters of the different generated data sets (n_j represents the number of the elements in cluster j).

	j	$\|\alpha_j\|$	μ_{j1}	μ_{j2}	μ_{j3}	$p(j)$	n_j
Data set 1	1	66	0.16	0.24	0.60	0.3	120
	2	105	0.22	0.48	0.30	0.3	120
	3	40	0.37	0.47	0.16	0.2	80
	4	92	0.31	0.09	0.60	0.2	80
Data set 2	1	66	0.16	0.24	0.60	0.2	150
	2	105	0.22	0.48	0.30	0.2	150
	3	40	0.37	0.47	0.16	0.2	150
	4	92	0.31	0.09	0.60	0.2	150
	5	116	0.52	0.34	0.14	0.2	150
Data set 3	1	66	0.16	0.24	0.60	0.2	200
	2	105	0.22	0.48	0.30	0.2	200
	3	40	0.37	0.47	0.16	0.2	200
	4	92	0.31	0.09	0.60	0.2	200
	5	116	0.52	0.34	0.14	0.1	100
	6	90	0.33	0.33	0.34	0.1	100
Data set 4	1	64	0.16	0.22	0.62	0.2	200
	2	105	0.22	0.48	0.30	0.2	200
	3	40	0.37	0.47	0.16	0.2	200
	4	92	0.31	0.09	0.60	0.1	100
	5	116	0.52	0.34	0.14	0.1	100
	6	90	0.33	0.33	0.34	0.1	100
	7	60	0.16	0.16	0.68	0.1	100

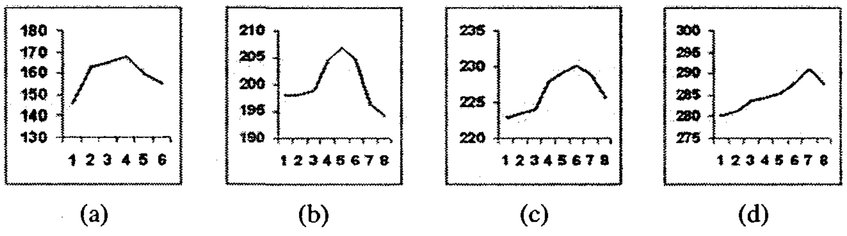

Figure 3.3: Number of clusters found for the different generated data sets using likelihood approach. (a) Data set 1, (b) Data set 2, (c) Data set 3, (d) Data set 4.Here X axis represents the number of clusters, and Y axis represents the MDL value.

A software is composed of a great number of relatively independent unites, performing certain functionalities, called modules. The classification of these modules into fault-prone and non fault-prone categories is important for software quality prediction and the identification of high-risk software components [15, 19, 80]. This classification is based on the description of the modules using software complexity metrics which have been shown to be related to the faults in a given software [140]. In this section we use our mixture model for the clustering of software modules. The data used in our experiments is taken from a software for a Medical Imaging System (MIS) widely used by the software

Table 3.2: Estimated parameters for the different generated data sets using both the likelihood and Bayesian approaches. $\hat{|\alpha_j|}^L$, $\hat{\mu}^L_{j1}$, $\hat{\mu}^L_{j2}$, $\hat{\mu}^L_{j3}$ and $\hat{p}(j)^L$ are the estimated parameters using the likelihood approach. $\hat{|\alpha_j|}^B$, $\hat{\mu}^B_{j1}$, $\hat{\mu}^B_{j2}$, $\hat{\mu}^B_{j3}$ and $\hat{p}(j)^B$ are the estimated parameters using the Bayesian approach.

	j	$\hat{\lvert\alpha_j\rvert}^L$	$\hat{\mu}^L_{j1}$	$\hat{\mu}^L_{j2}$	$\hat{\mu}^L_{j3}$	$\hat{p}(j)^L$	$\hat{\lvert\alpha_j\rvert}^B$	$\hat{\mu}^B_{j1}$	$\hat{\mu}^B_{j2}$	$\hat{\mu}^B_{j3}$	$\hat{p}(j)^B$
Set 1	1	65.21	0.14	0.25	0.61	0.28	65.07	0.17	0.26	0.57	0.27
	2	105.42	0.20	0.51	0.29	0.31	104.68	0.23	0.47	0.30	0.31
	3	41.23	0.35	0.48	0.17	0.18	41.75	0.38	0.44	0.18	0.21
	4	94.03	0.30	0.11	0.59	0.23	93.88	0.29	0.12	0.59	0.21
Set 2	1	66.19	0.17	0.24	0.59	0.19	66.34	0.16	0.26	0.58	0.19
	2	104.77	0.21	0.55	0.24	0.18	104.69	0.20	0.47	0.33	0.19
	3	40.28	0.36	0.46	0.18	0.2	39.56	0.38	0.47	0.15	0.2
	4	90.17	0.28	0.11	0.61	0.21	91.46	0.30	0.10	0.60	0.21
	5	118.02	0.50	0.35	0.15	0.22	117.66	0.51	0.36	0.13	0.21
Set 3	1	65.09	0.14	0.25	0.61	0.21	65.71	0.15	0.24	0.61	0.21
	2	105.56	0.24	0.47	0.29	0.18	104.78	0.21	0.47	0.32	0.19
	3	40.44	0.38	0.46	0.16	0.2	40.34	0.39	0.45	0.16	0.2
	4	91.12	0.30	0.10	0.60	0.2	92.18	0.31	0.10	0.59	0.2
	5	114.98	0.50	0.36	0.14	0.1	117.18	0.51	0.33	0.16	0.1
	6	90.94	0.32	0.34	0.34	0.11	88.96	0.35	0.34	0.31	0.1
Set 4	1	65.17	0.17	0.23	0.60	0.19	64.97	0.15	0.22	0.63	0.2
	2	107.19	0.20	0.49	0.31	0.19	104.01	0.21	0.50	0.29	0.19
	3	40.25	0.36	0.45	0.19	0.21	39.74	0.35	0.47	0.18	0.21
	4	91.20	0.30	0.08	0.62	0.11	92.88	0.30	0.10	0.60	0.1
	5	114.99	0.53	0.32	0.15	0.09	114.87	0.51	0.36	0.13	0.08
	6	90.77	0.32	0.31	0.37	0.08	89.80	0.32	0.34	0.34	0.1
	7	61.05	0.15	0.14	0.71	0.13	60.95	0.15	0.17	0.68	0.12

engineering community and can be found in [2]. More details about all these metrics can be found in [2, 31, 39]. Using these metrics each module was described by an 11-dimensional vector.

Our Bayesian and deterministic algorithms were applied to the 203 vectors representing the different modules. To monitor the convergence of the Bayesian algorithm, we run 5 parallel chains of 9000 iterations each. The values of the multiple potential scale reduction factor are shown in Fig. 3.5. According to this figure convergence occurs around the 7500 iteration.

Two types of errors can occur in our case. A Type I error when a non fault-prone module is classified as fault prone and a Type II error when a fault-prone is classified as non fault-prone. Table 3.3 shows the values of Type I and Type II errors when using both the deterministic and Bayesian algorithms. According to this table the two approaches give the same type I error which corresponds to 4 misclassified modules. The Bayesian approach outperforms the deterministic one in the case of type II error.

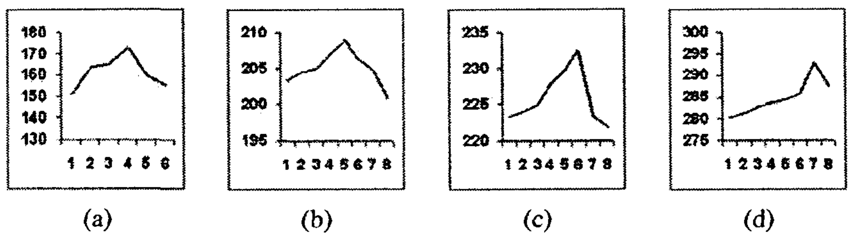

Figure 3.4: Number of clusters found for the different generated data sets using the Bayesian approach. (a) Data set 1, (b) Data set 2, (c) Data set 3, (d) Data set 4. Here X axis represents the number of clusters, and Y axis represents the MDL value.

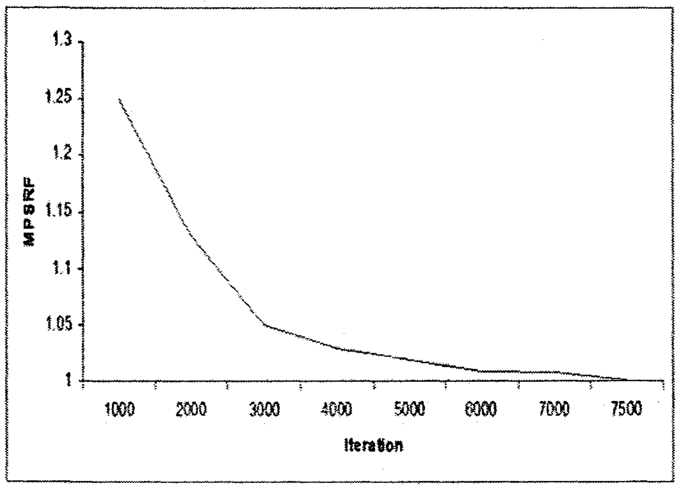

Figure 3.5: Plot of multiple potential scale reduction factor values.

Table 3.4 shows the classification probabilities of the 4 misclassified modules causing Type I errors. From this table, we can clearly see that the Bayesian approach has increased estimated probabilities, associated with the misclassified data samples, of belonging to the correct class (i.e non fault-prone).

Table 3.3: Type I and Type II errors using both the deterministic and Bayesian approaches.

	Type I error	Type II error
Maximum Likelihood	3.51%	28.08%
Bayesian	3.51%	26.96%

Table 3.4: Classification probabilities (probabilities to be in the non fault-prone class) of the misclassified modules causing type I errors.

Module Number	Bayesian	Maximum Likelihood
6	0.31	0.27
41	0.34	0.29
69	0.41	0.42
80 .	0.37	0.32

CHAPTER 4

Conclusion

In this thesis, we have studied the software modules classification problem which is a fundamental issue in software engineering, and different techniques were employed to accomplish it. Indeed, a lot of work has been devoted to locate high-risk modules in early software life-cycle by using modeling techniques including supervised and unsupervised learning approaches. These software quality prediction models can point out "hot spots" modules that are likely to have a high error rate or that need high development effort and further attention.

First, we performed a survey to compare and evaluate several selected modeling techniques using a real data set. The detailed experimental results and analysis were provided. Despite the success of many approaches, same problems still exist. An important problem is the choice of the number of metrics to describe a given module [53]. The description of the modules may include attributes based on subjective judgements which may give rise to errors in the values of the metrics. Besides, the collection of historical modules used for training may include some modules for which an incorrect classification was made. Another problem is the lack of sufficient data for learning in some cases.

Secondly, we proposed an unsupervised learning approach for software modules categorization. We present two algorithms for finite Dirichlet mixture estimation and selection. The first approach is deterministic based on maximum likelihood estimation using the EM algorithm. The second approach is purely Bayesian and is based on Gibbs sampling. Although the results obtained with these two approaches are frequently nearly similar, the approaches are conceptually different and can perform differently in some cases. The Bayesian approach is coherent and flexible compared to the deterministic one, but it has the disadvantage of being computationally expensive. From the experimental result, we can conclude that the finite Dirichlet mixture model successfully accomplish software categorization with relatively smaller misclassification rates. Although the achieved partial results in chapter 2 are better than what we got in chapter 3, the little discrepancy in these two sets of results does not deny the advantages of the finite Dirichlet mixture model, as a unsupervised learning technique which has no need of labelled collected training data sets, and possesses little inherent restrictions.

Further work can be devoted to the selection of the most relevant software complexity metrics for a given classification problem. In addition, the explanation of unsupervised algorithms for the complexity-based classification problem still needs more work. Another promising future work could be the use of semi-supervised approaches.

List of References

[1] P. Frankl, D. Hamlet, B. Littlewood and L. Strigini. Evaluating Testing Methods by Delivered Reliability. *IEEE Transactions on Software Engineering*, 24(8):586–601, 1998.

[2] J.C. Munson. *Handbook of Software Reliability Engineering*. IEEE Computer Society Press and McGraw-Hill Book Company, 1999.

[3] R. S. Pressman. *Software Engineering: A Practioner's Approach*. McGraw-Hill, New York, fifth edition, 2001.

[4] V. Y. Shen, T-J. Yu, S. M. Thebaut and L. R. Paulsen. Identifying Error-Prone Software- An Empirical Study. *IEEE Transactions on Software Engineering*, 11(4):317–324, 1985.

[5] A. A. Porter and R. W. Selby. Empirically guided software development using metric-based classification trees. *IEEE Software*, 7(2):46– 54, 1990.

[6] R. W. Selby. Empirically based analysis of failures in software systems. *IEEE Transactions on Reliability*, 39(4):444–454, 1990.

[7] T.M. Khoshgoftaar and E. B. Allen. Classification of Fault-Prone Software Modules: Prior Probabilities, Costs, and Model Evaluation. *Empirical Software Engineering*, 3(3):275–298, 1998.

[8] T. M. Khoshgoftaar and E.B. Allen. Early Quality Prediction: A Case Study in Telecommunications. *IEEE Software*, 13(4):65–71, 1996.

[9] T.M. Khoshgoftaar and E.B. Allen. A Practical Classification-Rule for Software-Quality Models. *IEEE Transactions on Reliability*, 49(2):209–216, 2000.

[10] T. M. Khoshgoftaar, D. L. Lanning and A. S. Pandya. A Comparative Study of Pattern Recognition Techniques for Quality Evaluation of Telecommunications Software. *IEEE Journal on Selected Areas in Communications*, 12(2):279–291, 1994.

[11] S. Henry and S. Wake. Predicting maintainability with software quality metrics. *Journal of Software Maintenance: Research and Practice*, 3(3):129–143, 1991.

[12] L. C. Briand, V. R. Basili and C. J. Hetmanski. Developing Interpretable Models with Optimized Set Reduction for Identifying High-Risk Software Components. *IEEE Transactions on Software Engineering*, 19(11):1028–1044, 1993.

[13] G. W. Russel. Experience With Inspection in Ultralarge-Scale Developments. *IEEE Software*, 8(1):25–31, 1991.

[14] T.M. Khoshgoftaar, E. B. Allen, R. Halstead and G.P. Trio. Detection of fault-prone software modules during a spiral life cycle. In *Proc. of International Conference on Software Maintenance*, pages 69–76, 1996.

[15] J.C. Munson and T.M. Khoshgoftaar. The Detection of Fault-Prone Programs. *IEEE Transactions on Software Engineering*, 18(5):423–433, 1992.

[16] S. G. Stockman, A. R. Todd, and G. A. Robinson. A Framework for Software Quality Measurement. *IEEE Journal on Selected Areas in Communications*, 8(2):224–233, 1990.

[17] J. Henry, S. Henry, D. Kafura and L. Matheson. Improving Software Maintenance at Martin Marietta. *IEEE Software*, 11(4):67–75, 1994.

[18] T.M. Khoshgoftaar, E. B. Allen, W. D. Jones and J. P. Hudepohl. Which Software Modules have Faults which will be Discovered by Customers? *Journal of Software Maintenance: Research and Practice*, 11:1–18, 1999.

[19] L. C. Briand, V. R. Basili and W. M. Thomas. A Pattern Recognition Approach for Software Engineering Data Analysis. *IEEE Transactions on Software Engineering*, 18(11):931–942, 1992.

[20] R. W. Selby and A. A. Porter. Learning From Examples: Generation and Evaluation of Decision Trees for Software Ressource Analysis. *IEEE Transactions on Software Engineering*, 14(12):1743–1757, 1988.

[21] T. M. Khoshgoftaar and D. L. Lanning. A neural network approach for early detection of program modules having high risk in the maintenance phase. *Journal of Systems and Software*, 29(1):85–91, 1995.

[22] T.M. Khoshgoftaar, A.S. Pandya and D. L. Lanning. Application of Neural Networks for Predicting Program Faults. *Annals of Software Engineering*, 1(1):141–154, 1995.

[23] N. F. Schneidewind. Software metrics validation: Space Shuttle flight software example. *Annals of Software Engineering*, 1(1):287–309, 1995.

[24] C. Ebert. Classification Techniques for Metric-Based Development. *Software Quality Journal*, 5(4):255–272, 1996.

[25] T.M. Khoshgoftaar, E.B. Allen, L.A. Bullard, R. Halstead, G.P. Trio. A tree-based classification model for analysis of a military software system. In *Proc. of High-Assurance Systems Engineering Workshop*, pages 244–251, 1996.

[26] T.M. Khoshgoftaar and E. B. Allen. Modeling fault-prone modules of subsystems. In *Proc. of the 11th International Symposium on Software Reliability Engineering*, pages 259–267, 2000.

[27] T.M. Khoshgoftaar, X. Yuan, E. B. Allen, W. D. Jones and J. P. Hudepohl. Uncertain Classification of Fault-Prone Software Modules. *Empirical Software Engineering*, 7(1):297–318, 2002.

[28] S. S. Gokhale and M. R. Lyu. Regression Tree Modeling for the Prediction of Software Quality. In *Proc. of the third ISSAT International Conference on Reliability and Quality in Design*, pages 31–36, 1997.

[29] T.M. Khoshgoftaar and J.C. Munson. Predicting Software Development Errors Using Software Complexity Metrics. *IEEE Journal on Selected Areas in Communications*, 8(2):253–261, 1990.

[30] G. Le Gall, M.-F. Adam, H. Derriennic, B. Moreau and N. Valette. Studies on Measuring Software. *IEEE Journal on Selected Areas in Communications*, 8(2):234–246, 1990.

[31] T.J. McCabe. A Complexity Measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, 1976.

[32] L. Briand, K. EL Emam and S. Morasca. On the Application of Measurement Theory in Software Engineering. *Empirical Software Engineering*, 1(1):61–88, 1996.

[33] H. Zuse. Comments to the Paper: Briand, Eman, Morasca: On the Application of Measurement Theory in Software Engineering. *Empirical Software Engineering*, 2(3):313–316, 1997.

[34] N. Fenton. Software Measurement: A Necessary Scientific Basis. *IEEE Transactions on Software Engineering*, 20(3):199–206, 1994.

[35] S. L. Pfleeger, J. C. Fitzgerald and D. A. Rippy. Using multiple metrics for analysis of improvement. *Software Quality Journal*, 1(1):27–36, 1992.

[36] S. L. Pfleeger. Lessons Learned in Building a Corporate Metrics Program. *IEEE Software*, 10(3):67–74, 1993.

[37] B. Curtis, S. B. Sheprad, H. Milliman, M. A. Borst and T. Love. Measuring the Psychlogical Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics. *IEEE Transactions on Software Engineering*, SE-5(2):96–104, 1979.

[38] L. M. Ottenstein. Quantitative Estimates of Debugging Requirements. *IEEE Transactions on Software Engineering*, SE-5(5):504–514, 1979.

[39] H. Jensen and K. Vairavan. An Experimental Study of Software Metrics for Real-time Software. *IEEE Transaction on Software Engineering*, SE-11(4):231–234, 1994.

[40] D. L. Lanning and T.M. Khoshgoftaar. Fault Severity in Models of Fault-Correction Activity. *IEEE Transactions on Reliability*, 44(4):666–671, 1995.

[41] S. D. Conte. *Metrics and Models in Software Quality Engineering*. Addison-Wesley Professional, 1996.

[42] W. Li and S. Henry. Object-Oriented Metrics that Predict Maintainability. *Journal of Systems and Software*, 23(2):111–122, 1993.

[43] E.J. Weyuker. Evaluating software complexity measures. *IEEE Transactions on Software Engineering*, 14(9):1357–1365, 1988.

[44] M. H. Halstead and A.M. Leroy. *Elements of Software Science*. New York: Elseviser, 1977.

[45] V. Y. Shen, S. D. Conte and H. E. Dunsmore. Software Science Revisited: A Critical Analysis of the Theory and its Empirical Support. *IEEE Transactions on Software Engineering*, SE-9(2):155–165, 1983.

[46] V. R. Basili, L. C. Briand and W. L. Melo. A Validation of Object-Oriented Design Metrics as Quality Indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, 1996.

[47] L. Mark and K. Jeff. *Object-Oriented Software Metrics.* Prentice-Hall, 1994.

[48] R. M. Szabo and T.M. Khoshgoftaar. An assessment of software quality in a C++ environment. In *Proc. of the Sixth International Symposium on Software Reliability Engineering*, pages 240–249, 1995.

[49] S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering* , 20(6):476–493, 1994.

[50] M. Hitz and B. Montazeri. Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective. *IEEE Transactions on Software Engineering* , 22(4):267–271, 1996.

[51] T. M. Khoshgoftaar, J. C. Munson and D. L. Lanning. Alternative approaches for the use of metrics to order programs by complexity. *Journal of Systems and Software*, 24(3):211–221, 1994.

[52] A. Mayer, A. M. Sykes. Statistical Methods for the Analysis of Software Metrics Data. *Software Quality Journal*, 1(4):209–223, 1992.

[53] N. F. Schneidewind. Software metrics model for integrating quality control and prediction. In *Proc. of the Eighth International Symposium on Software Reliability Engineering*, pages 402–415, 1997.

[54] On Statistics in Software Engineering Measurement. A Composite Complexity Approach for Software Defect Modeling. *Software Quality Journal*, 2(1):49–60, 1993.

[55] N. Ohlisson, M. Zhao and M. Helander. Application of Multivariate Analysis for Software Fault Prediction. *Software Quality Journal*, 7(1):51–66, 1998.

[56] M. C. Ohlsson and C. Wohlin. Identification of Green, Yellow and Red Legacy Components. In *Proc. of the International Conference on Software Maintenance*, pages 6–15, 1998.

[57] N. F. Schneidewind. Methodology For Validating Software Metrics. *IEEE Transactions on Software Engineering*, 18(5):410–422, 1992.

[58] N. F. Schneidewind. Minimizing risk in applying metrics on multiple projects. In *Proc. of Third International Symposium on Software Reliability Engineering*, pages 173–182, 1992.

[59] B.D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[60] J. C. Munson and T. M. Khoshgoftaar. The Dimensionality of Program Complexity. In *Proc. of Eleventh International Conference on Software Engineering*, pages 245–253, 1989.

[61] T.M. Khoshgoftaar and E. B. Allen. Multivariate Assessment of Complex Software Systems: A comparative Study. In *Proc. of First International Conference on Engineering of Complex Computer Systems*, pages 389–396, 1995.

[62] T.M. Khoshgoftaar, E. B. Allen and N. Goel. The Impact of Software Evolution and Reuse on Software Quality. *Empirical Software Engineering*, 1(1):31–44, 1996.

[63] J. C. Munson and T. M. Khoshgoftaar. Improving Tree-Based Models of Software Quality with Principal Components Analysis. In *Proc. of 11th International Symposium on Software Reliability Engineering*, pages 198–209, 2000.

[64] R. O. Duda, P. E. Hart and D. G. Stork. *Pattern Classification*. Wiley, New York, 2001.

[65] W. R. Dillon and M. Goldstein. *Multivariate Analysis*. New York: Wiley, 1984.

[66] V. Rodriguez and W. T. Tsai. Evaluation of Software Metrics Using Discriminant Analysis. *Information and Software Technology*, 29(3):245–251, 1987.

[67] N. F. Schneidewind. Validating Software Metrics: Producing Quality Discriminators. In *Proc. of Second International Symposium on Software Reliability Engineering*, pages 225–232, 1991.

[68] T.M. Khoshgoftaar and R. Halstead. Process Measures for Predicting Software Quality. In *Proc. of High-Assurance Systems Engineering Workshop*, pages 155–160, 1997.

[69] T.M. Khoshgoftaar and E. B. Allen. The Impact of Costs of Misclassification on Software Quality Modeling. In *Proc. of Fourth International Software Metrics Symposium*, pages 54–62, 1997.

[70] D. L. Lanning and T.M. Khoshgoftaar. The impact of software enhancement on software reliability. *IEEE Transactions on Reliability*, 44(4):677–682, 1995.

[71] T.M. Khoshgoftaar and E. B. Allen. A Comparative Study of Ordering and Classification of Fault-Prone Software Modules. *Empirical Software Engineering*, 4(2):159–186, 1999.

[72] D. C. Montgomery, E. A. Peck and G. G. Vining. *Introduction to Linear Regression Analysis*. Wiley-Interscience, third edition, 2001.

[73] T. M. Khoshgoftaar, J. C. Munson, B. B. Bhattacharya and G. D. Richardson. Predictive Modeling Techniques of Software Quality from Software Measures. *IEEE Transactions on Software Engineering*, 18(11):979–987, 1992.

[74] T. M. Khoshgoftaar, J. C. Munson and D. L. Lanning. A comparative Study of Predictive Models for Program Changes During System Testing and Maintenance. In *Proc. of the IEEE Conference on Software Maintenance*, pages 72–79, 1993.

[75] D. W. Hosmer and S. Lemeshow. *Applied Logistic Regression.* Wiley-Interscience Publication, 2000.

[76] L. C. Briand, W. M. Thomas and C. J. Hetmanski. Modeling and Managing Risk Early in Software Development. In *Proc. of 15th International Conference on Software Engineering*, pages 55–65, 1993.

[77] T.M. Khoshgoftaar and E. B. Allen. Logistic Regression modeling of Software Quality. *International Journal of Reliability, Quality and Safety*, 6(4):303–317, 1999.

[78] N. F. Schneidewind. Investigation of Logistic Regression as a Discriminant of Software Quality. In *Proc. of the Seventh IEEE Symposium on Software Metrics*, pages 328–337, 2001.

[79] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, second edition, 1999.

[80] F. Xing, P. Guo and M.R. Lyu. A Novel Method for Early Software Quality Prediction Based on Support Vector Machine. In *Proc. of the 16th IEEE International Symposium on Software Reliability Engineering*, pages 213–222, 2005.

[81] G.J. McLachlan and D. Peel. *Finite Mixture Models.* New York: Wiley, 2000.

[82] P.Guo and M.R. Lyu. Software Quality Prediction Using Mixture Models with EM Algorithm. In *In Proc. First Asia-Pacific Conference on Quality Software*, pages 69–78, 2000.

[83] G. Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464, 1978.

[84] C. S. Wallace. *Statistical and Inductive Inference by Minimum Message Length.* Springer, 2005.

[85] H. Akaike. A New Look at the Statistical Model Identification. *IEEE Transactions on Automatic Control*, AC-19(6):716–723, 1974.

[86] J. Rissanen. Modeling by Shortest Data Description. *Automatica*, 14:465–471, 1978.

[87] R. Takahashi, Y. Muraoka and Y. Nakamura. Building Software Quality Classification Trees: Approach, Experimentation, Evaluation. In *Proc. of the 8th IEEE International Symposium on Software Reliability Engineering*, pages 222–233, 1997.

[88] T.G. Dietterich. Approximate Statistical Test For Comparing Supervised Classification Learning Algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

[89] M. Shepperd and G. Kadoda. Comparing Software Prediction Techniques Using Simulation. *IEEE Transactions on Software Engineering*, 27(11):1014–1022, 2001.

[90] N.Bouguila, J.H.Wang and A.Ben.Hamza. A Bayesian Approach for Software Quality Prediction. *IEEE International Conference on Intelligent Systems*, 2008. accepted.

[91] N.Bouguila, J.H.Wang and A.Ben.Hamza. Software Modules Categorization Through Likelihood and Bayesian Analysis of Finite Dirichlet Mixtures. *Journal of Applied Statistics*, 2008. submitted.

[92] P. Giudici. *Applied Data Mining: Statistical Methods for Business and Industry.* Wiley, 2003.

[93] B.S. Everitt and D.J. Hand. *Finite Mixture Distributions.* Chapman and Hall, London, UK., 1981.

[94] D.M. Titterington, A.F.M. Smith and U.E. Markov. *Statistical Analysis of Finite Mixture Distributions.* New York: Wiley, 1985.

[95] K. Pearson. Contributions to the Theory of Mathematical Evolution. *Philosophical Transactions of the Royal Society of London*, A 185:71–110, 1894.

[96] N. Bouguila, D. Ziou and E. Monga. Practical Bayesian Estimation of a Finite Beta Mixture Through Gibbs Sampling and its Applications. *Statistics and Computing*, 16(2):215–225, 2006.

[97] I. R. James and J. E. Mosimann. A New Characterization of the Dirichlet Distribution Through Neutrality. *The Annals of Statistics*, 8(1):183–189, 1980.

[98] J. Fabius. Two Characterizations of the Dirichlet Distribution. *The Annals of Statistics*, 1(3):583–587, 1973.

[99] J. N. Darroch and D. Ratcliff. A Characterization of the Dirichlet Distribution. *Journal of the American Statistical Association*, 66(335):641–643, 1971.

[100] B. V. Rao and B. K. Sinha. A Characterization of Dirichlet Distributions. *Journal of Multivariate Analysis*, 25:25–30, 1988.

[101] J. F. Chamayou and G. Letac. A Transient Random Walk on Stochastic Matrices with Dirichlet Distributions. *The Annals of Probability*, 22(1):424–430, 1994.

[102] D. Geiger and D. Heckerman. A Characterization of the Dirichlet Distribution Through Global and Local Independence. *The Annals of Statistics*, 25:1344–1369, 1997.

[103] G. Letac and H. Massam. A Formula on Multivariate Dirichlet Distributions. *Statistics & Probability Letters*, 38(3):247–253, 1998.

[104] A. Narayanan. A Note on Parameter Estimation in the Multivariate Beta Distribution. *Computer Mathematics and Applications*, 24(10):11–17, 1992.

[105] G. Ronning. Maximum Likelihood Estimation of Dirichlet Distributions. *Journal of Statistical Computation and Simulation*, 32:215–221, 1989.

[106] J. E. Mosimann. On the Compound Multinomial Distribution, the Multivariate β-Distribution, and Correlations Among Proportions. *Biometrika*, 49(1 and 2):65–82, 1962.

[107] D. J. C. Mackay and L. Peto. A Hierarchical Dirichlet Language Model. *Natural Language Engineering*, 1(3):1–19, 1994.

[108] N. Bouguila and D. Ziou. Unsupervised Learning of a Finite Discrete Mixture: Applications to Texture Modeling and Image Databases Summarization. *Journal of Visual Communication and Image Representation*, 18(4):295–309, 2007.

[109] N. Bouguila, D. Ziou and J. Vaillancourt. Novel Mixtures Based on the Dirichlet Distribution: Application to Data and Image Classification. In *Machine Learning and Data Mining in Pattern Recognition (MLDM)*, pages 172–181, Leipzig, Germany, 2003. Springer, LNAI2734.

[110] N. Bouguila, D. Ziou and J. Vaillancourt. Unsupervised Learning of a Finite Mixture Model Based on the Dirichlet Distribution and its Application. *IEEE Transactions on Image Processing*, 13(11):1533–1543, 2004.

[111] N. Bouguila and D. Ziou. Unsupervised Selection of a Finite Dirichlet Mixture Model: An MML-Based Approach. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):993–1009, 2006.

[112] G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. New York: Wiley-Interscience, 1997.

[113] A.P. Dempster, N.M. Laird and D.B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society, B*, 39:1–38, 1977.

[114] X. Meng and D. van Dyk. The EM Algorithm - An Old Folk Song Sung to a Fast New Tune . *Journal of the Royal Statistical Society, B*, 59(3):511–567, 1997.

[115] W. M. Bolstad. *Introduction to Bayesian Statistics*. John Wiley and Sons, 2004.

[116] P. M. Lee. *Bayesian Statistics: An Introduction*. Arnold Publication, third edition, 2004.

[117] M. Aitkin. Likelihood and Bayesian Analysis of Mixtures. *Statistical Modelling*, 1:287–304, 2001.

[118] J.M. Marin, K. Mengersen and C.P. Robert. Bayesian modeling and inference on mixtures of distributions. In D. Dey and C.R. Rao, editors, *Handbook of Statistics 25*. Elsevier-Sciences, 2004.

[119] C.E. Rasmussen. The Infinite Gaussian Mixture Model. In T.K. Leen S.A. Solla and K.-R. Müller, editors, *Advances in Neural Information Processing Systems (NIPS)*, pages 554–560. MIT Press, 2000.

[120] N. Metropolis and S. Ulam. The Monte Carlo Method. *Journal of the American Statistical Association*, 44:335–341, 1949.

[121] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller. Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[122] W. K. Hastings. Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika*, 57:97–109, 1970.

[123] G. Casella, K. Mengersen, C. Robert and D. Titterington. Perfect Slice Samplers for Mixtures of Distributions. *Journal of the Royal Statistical Society, B*, 64(4):777–790, 2000.

[124] A. E. Raftery and S. Lewis. How Many Iterations in Gibbs Sampler? In A. P. Dawid J. Bernardo, J. O. Berger and A. F. M. Smith, editors, *Bayesian Statistics 4*, pages 763–773. Oxford University Press, 1992.

[125] B. D. Ripley. *Stochastic Simulation*. Wiley, New York, 1987.

[126] M. A. Tanner and W. H. Wong. The Calculation of Posterior Distributions by Data Augmentation (With Discussion). *Journal of the American Statistical Association*, 82:528–550, 1987.

[127] A. E. Gelfand and A. F. M. Smith. Sampling-Based Approaches to Calculating Marginal Densities. *Journal of the American Statistical Association*, 85:398–409, 1990.

[128] A. Gelman and D. B. Rubin. Inference From Iterative Simulation Using Multiple Sequences. *Statistical Science*, 7(4):457–472, 1992.

[129] S. P. Brooks and A. Gelman. General Methods for Monitoring Convergence of Iterative Simulations. *Journal of Computational and Graphical Statistics*, 7:434–455, 1998.

[130] P. Congdon. *Applied Bayesian Modelling*. John Wiley and Sons, 2003.

[131] I. L. Tsung, C. L. Jack and F. N. Huey. Bayesian Analysis of Mixture Modeling Using the Multivariate t Distribution. *Statistics and Computing*, 14:119–130, 2004.

[132] D. V. Lindley. Approximate Bayesian Methods. In D. V. Lindley J. Bernardo, M. DeGroot and A. F. M. Smith, editors, *Bayesian Statistics*, pages 223–237. Valencia University Press, 1980.

[133] R.E. Kass and A.E. Raftery. Bayes Factors. *Journal of the American Statistical Association*, 90:773–795, 1995.

[134] D. M. Chickering and D. Heckerman. Efficient Approximations for the Marginal Likelihood of Bayesian Networks With Hidden Variables. *Machine Learning*, 29:181–212, 1997.

[135] A. E. Raftery. Hypothesis testing and model selection. In D.J. Spiegelhalter W.R. Gilks and S. Richardson, editors, *Markov Chain Monte Carlo in Practice*, pages 163–188. London: Chapman and Hall, 1996.

[136] R. Kass, L. Tierney and J. Kadane. Asymptotics in Bayesian Computation. In D. V. Lindley J. Bernardo, M. DeGroot and A. F. M. Smith, editors, *Bayesian Statistics 3*, pages 261–278. Oxford University Press, 1988.

[137] G. Schwarz. Estimating the Dimension of a Model. *Annals of Statistics*, 6:461–464, 1978.

[138] V. Y. Shen, T. J. Yu, S. M. Thebaut and L. R. Paulsen. Identifying Error-Prone Software - An Empirical Study. *IEEE Transactions on Software Engineering*, 11(4):317–324, 1985.

[139] L. C. Briand, V. R. Basili and W. M. Thomas. A Pattern Recognition Approach for Software Engineering Data Analysis. *IEEE Transactions on Software Engineering*, 18(11):931–942, 1992.

[140] T. Khoshgoftaar and J. Munson. Predicting Software Development Error Using Software Complexity Metrics. *IEEE Transactions on Software Engineering*, 8(2):253–261, 1990.