Software Quality Prediction Using Mixture Models with EM Algorithm*

Ping Guo and Michael R. Lyu
Department of Computer Science & Engineering,
The Chinese University of Hong Kong,
Shatin, NT, Hong Kong
Email: pguo@cse.cuhk.edu.hk, lyu@cse.cuhk.edu.hk

Abstract

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

1 Introduction

Software reliability engineering is one of the most important aspect of software quality [1]. The interest of the software community in program testing continues to grow – as does the demand for complex, and predictively reliable programs. It is no longer acceptable to postpone the assurance of software quality until prior to a product's release. Delaying corrections until testing and operational phases may lead to higher costs [2], and it may be too late to improve the system significantly. Recent research in the field of computer program reliability has been directed towards the identification of software modules that are likely to be fault–prone, based on product and/or process–related metrics, prior to the testing phase, so that early identification of fault–prone modules in the life–cycle can help in channeling program testing and verification efforts in the productive direction.

Software metrics represent quantitative description of program attributes and the critical role they play in predicting the quality of the software has been emphasized by Perlis *et al* [3]. That is, there is a direct relationship between some complexity metrics and the number of changes attributed to faults later found in test and validation [4]. Many researchers have sought to develop a predictive relationship between complexity metrics and faults. Crawford *et al* [5] suggest that multiple variable models are necessary to find metrics that are important in addition to program size. Consequently, investigating the relationship between the number of faults in programs and the software complexity metrics attracts researchers' interesting.

Several different techniques have been proposed to develop predictive software metrics for the classification of software program modules into fault–prone and non fault–prone categories. These techniques include discriminant analysis [6, 7], factor analysis [8], classification trees [9, 10], pattern recognition (Optimal Set Reduction (OSR)) [6, 11], feedforward neural networks [12], and some other techniques [13]. Most of these techniques are classification models and they partition the modules into two categories, namely, fault–prone and not fault–prone. With these predictive models, the troublesome modules can be identified earlier in the life–cycle of a software product. The advantage of these fault prediction models are multi-fold; however, when building the models, they require to know the number of changes (faults) at the same time. That is, we have to know the target value first to build the model, using neural network terminology to describe this – the model parameters need to be estimated with a supervised learning procedure [14]. As we know, to obtain the dependent criterion variable, we will need to a long time for the feedback of test and validation results. For example, for the software of Medical Imaging System (MIS) presented later in this paper, the actual number of changes (faults) in that program is collected during three-year observation period. As software complexity metrics can be obtained relatively early in the software life-cycle, it is worthy to explore new techniques for early prediction of software quality based on software

*This work is supported by the grant of CUHK4432/99E.

0-7695-0825-1/00 $10.00 © 2000 IEEE

complexity metrics.

In this paper we present one such new approach – using a finite mixture model with Expectation-Maximum (EM) algorithm [15, 16] to investigate the predictive relationship between software metrics and the classification of the program module. With the mixture model analysis, we can develop a prediction model without the need to know the number of changes (faults) in advance. Namely, it is only based on software complexity metrics to build the model. The model parameters are estimated by using EM algorithm, which is a procedure of unsupervised learning since the class membership of those metrics is unknown and the metrics are treated as un-labeled vectors.

The mixture model analysis is mainly a probabilistic classification procedure. It is used to assign program modules to classes of modules of similar characteristics without the knowledge of fault rate in advance. By this statistical technique, we can identify a program or a program module as a class of low or high fault rate in the early stage of program development. In addition, we also show that the discriminant analysis is a special case of the mixture model analysis.

2 Modeling Methodology

We propose to use the finite mixture model analysis with EM algorithm technique in software quality prediction to classify fault-prone and non fault-prone modules. In the following we will briefly review the mixture model with EM algorithm, and Akaike Information Criterion (AIC) model selection criterion.

The mixture distribution, particular in Gaussian (normal) analysis method, has been used widely in a variety of important practical situations, where the likelihood approach to the fitting of mixture models has been utilized extensively [17, 18, 19, 20]. The application of the finite mixture model to software quality prediction is based on the assumption that the software complexity metrics in a vector space can be considered as a sample arising from two or more models mixed in varying proportions.

2.1 Finite Gaussian Mixture Model With EM Algorithm

A mixture model can be of any mixed distribution function, but the mostly-used model is the Gaussian distribution model. Hence, in this paper we only investigate the Gaussian density case. In the software complexity metrics vector space, one module can be considered as one point, and altogether N points consistent of N modules can form a given data set D. The data set $D = \{\mathbf{x}_i\}_{i=1}^N$ ready for classification is assumed to be samples from a mixture of k Gaussian densities with joint probability density

$$p(\mathbf{x}, \Theta) = \sum_{j=1}^{k} \alpha_j G(\mathbf{x}, \mathbf{m}_j, \mathbf{\Sigma}_j),$$
$$\text{with} \quad \alpha_j \geq 0, \text{ and } \sum_{j=1}^{k} \alpha_j = 1 \qquad (1)$$

where

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

is multivariate Gaussian density function, $\mathbf{x}$ denotes random vector (which integrates a variety of software metrics), d is the dimension of $\mathbf{x}$, and parameter $\Theta = \{\alpha_j, \mathbf{m}_j, \mathbf{\Sigma}_j\}_{j=1}^k$ is a set of finite mixture model parameter vectors. Here α_j is the mixing weights, $\mathbf{m}_j$ is the mean vector, and $\mathbf{\Sigma}_j$ is the covariance matrix of the j-th component. In fact, as these parameters are unknown, using how many Gaussian density components can best describe the probability density of the system is also unknown. Usually with a pre-assumed number k, the mixture model parameters are estimated by the maximum likelihood learning (ML) with EM algorithm [15, 16].

The log likelihood function of the system to be explored is

$$l(\Theta|x) = \ln L(\Theta|x) = \sum_{i=1}^{N} \ln(\sum_{j=1}^{k} \alpha_j G(\mathbf{x}_i, \mathbf{m}_j, \mathbf{\Sigma}_j)) \qquad (3)$$

Maximizing this function will re-derive the EM algorithm, which we show in two steps.

1. **E-step:**(Expectation step)

Calculate the *posterior* probability $p(j|\mathbf{x}_i)$ according to

$$p(j|\mathbf{x}) = \frac{\alpha_j G(\mathbf{x}, \mathbf{m}_j, \widehat{\mathbf{\Sigma}}_j)}{p(x, \Theta)}, \qquad \text{with } j = 1, 2, \cdots, k, \qquad (4)$$

2. **M-step:**(Maximum step)

$$\alpha_j^{new} = \frac{1}{N} \sum_{i=1}^{N} \frac{\alpha_j^{old} G(\mathbf{x}_i, \mathbf{m}_j, \mathbf{\Sigma}_j)}{\sum_{j=1}^{k} \alpha_j^{old} G(\mathbf{x}_i, \mathbf{m}_j, \mathbf{\Sigma}_j)} = \frac{1}{N} \sum_{i=1}^{N} p(j|\mathbf{x}_i) \qquad (5)$$

$$\mathbf{m}_j = \frac{\sum_{i=1}^{N} p(j|\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1}^{N} p(j|\mathbf{x}_i)} = \frac{1}{\alpha_j N} \sum_{i=1}^{N} p(j|\mathbf{x}_i)\mathbf{x}_i \qquad (6)$$

$$\widehat{\mathbf{\Sigma}}_j = \frac{1}{\alpha_j N} \sum_{i=1}^{N} p(j|\mathbf{x}_i)[(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T]. \qquad (7)$$

The two steps are iterated until convergence to one local minima is obtained.

Unlike supervised learning, the ML with EM algorithm can be used for a totally un-labeled data set; that is, the case of sample class membership is unknown.

In practical implementation, the problem to be handled first is the mixture parameter initialization. It is a common practice that the parameter values are random initialized since no *a priori* information is available. In this paper, we use the following methods to initialize mixture model parameters:

$$\alpha_j^0 = \frac{1}{k}, \tag{8}$$

$$\mathbf{m}_j^0 = \min_{1\le i\le N}(\mathbf{x}_i) + j \times \{\max_{1\le i\le N}(\mathbf{x}_i) - \min_{1\le i\le N}(\mathbf{x}_i)\}/(k+1) \tag{9}$$

$$\widehat{\mathbf{\Sigma}}_j^0 = \frac{||\max(\mathbf{x}_i) - \min(\mathbf{x}_i)||}{20}\mathbf{I}_d. \tag{10}$$

where $\mathbf{I}_d$ represents the $d \times d$ dimension identity matrix. This initialization method can guarantee that the mean vectors are within the range of the data set D. The alternative method used is an addition of a small random value on the above equations.

2.2 Model Selection Criterion

When the software complexity metric data are to be classified into several classes, each class contain the data samples with similar characteristics. With prior knowledge, we usually divide the modules into two classes: one is fault-prone and the other is non fault-prone. However, by the mixture model approach, how many classes the metric data should be divided is not known. Consequently, the number of Gaussian density components can best describe the probability density of the system is unknown. Nevertheless, we can use some model selection criterion to determine a proper number of model components.

Following Akaike's pioneering work [21] in selecting the number of components in the mixture model analysis, a lot of researchers have developed some modified and newly proposed criteria such as AICB [22], CAIC [23], SIC [24]. These criteria combine the maximum value of the likelihood function with the number of parameters used in achieving that value. Here we list the corresponding AIC formula for a convenient use afterwards, in which $L(k)$ means likelihood function of the number k model with other parameters like Θ has been estimated by using the equation (3):

$$AIC(k) = -2\ln[\max L(k)] + 2m_k, \tag{11}$$

where the $m_k = kd + (k-1) + kd(d+1)/2$ is a penalty term. The other criteria such as AICB, CAIC and SIC are similar to AIC, with the difference at the penalty term.

From the above $AIC(k)$, we can select the model number k^* simply by $k^* = \arg\min_k AIC(k)$ with ML obtained parameter Θ^*. In practice, we start with $k = 1$, estimate parameter Θ^*, and compute $AIC(k=1)$. Then by iterating $k \to k+1$, we compute $AIC(k=2)$, and so on. After getting a series of $AIC(k)$, we choose the minimal one and get the corresponding k^*. This k^* is assumed as the number of classes of the program modules should be partitioned.

2.3 Bayesian Probabilistic Classification

In the mixture model case a Bayesian decision rule is used to classify the vector $\mathbf{x}$ into class j with the largest *posterior* probability. The *posterior* probability $p(j|\mathbf{x})$ represents the probability that sample $\mathbf{x}$ belongs to class j. The probabilities of $p(j|\mathbf{x})$ are usually unknown and have to be estimated from the training samples. With the maximum likelihood estimation, the *posterior* probability can be written in the form of equation (4).

For a given $\mathbf{x}_i$, we can obtain k probabilities $p(j=1|\mathbf{x}_i)$, $p(j=2|\mathbf{x}_i)$, $\cdots$, $p(j=k|\mathbf{x}_i)$. Now we use the Bayesian decision rule to classify $\mathbf{x}_i$ into one of the non-overlapping class j^* by the solution of

$$j^* = \arg\max_j p(j|\mathbf{x}_i), \qquad \text{for } j = 1, 2, \cdots, k. \tag{12}$$

If j^* is corresponding to maximum $p(j|\mathbf{x}_i)$, the ith program module will be classified into class j^* with probability $p(j^*|\mathbf{x}_i)$.

When we take the logarithm to equation (4) and omit the common factors of the classes, such as $\ln p(x, \Theta)$, $d/2\ln 2\pi$, the classification rule becomes

$$j^* = \arg\min_j d_j(\mathbf{x}), \qquad \text{for } j = 1, 2, \cdots, k \tag{13}$$

with

$$d_j(\mathbf{x}) = (\mathbf{x}-\mathbf{m}_j)^T\mathbf{\Sigma}_j^{-1}(\mathbf{x}-\mathbf{m}_j) + \ln|\mathbf{\Sigma}_j| - 2\ln\alpha_j \tag{14}$$

This equation is often called the discriminant score for the jth class in the literature [25]. Furthermore, if the *prior* density α_j is the same for all classes (an equal sample number in each class), it becomes discriminant function when omitting the term $2\ln\alpha_j$. If a pooled covariance matrix is used, it is called linear discriminant analysis (LDA), which was used by Munson and Khoshgoftaar for detection of fault-prone programs [7].

If the class membership relation of the sample as well as the number N_j of each class is known, which is assumed in the discriminant analysis application [7], the mean vector $\mathbf{m}_j$ and the covariance matrix $\mathbf{\Sigma}_j$ can be evaluated based on given samples with maximum likelihood estimation. They take the following forms:

$$\mathbf{m}_j = \frac{1}{N_j}\sum_{i=1}^{N_j}\mathbf{x}_i \qquad (15)$$

$$\widehat{\Sigma}_j = \frac{1}{N_j - 1}\sum_{i=1}^{N_j}(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T. \qquad (16)$$

They are called sample mean and sample covariance matrix, respectively [26]. Here we can see they are different with EM estimate. In a supervised learning case, each sample has determined class membership, while in EM estimate, each sample can belong to every class at the same time with a certain probability value.

3 Data Description and Analysis Procedure

In this section, we present a real project to which we apply the finite mixture model with EM algorithm for quality prediction and data analysis. The data used for the application of the mixture model represents the results of an investigation of software for a Medical Imaging System (MIS). The total system consisted of about 4500 modules amounting to about 400,000 lines of code written in Pascal, FORTRAN, assembler and PL/M. A random sample of 390 modules, from the ones written in Pascal and FORTRAN were selected for analysis. These 390 modules consists of approximately 40,000 lines of code. The software was developed over a period of five years, and was in commercial use at several hundred sites for a period of three years[12].

The number of changes made to a module, documented as Change Reports (CRs), was used as an indicator of the number of faults introduced during development[27]. The changes made to the routines were analyzed, and only those that affected the executable code were counted as faults (aesthetic changes such as comments were not counted)[28].

In addition to the change data, the following 11 software complexity metrics were developed for each of the modules:

- Total lines of code (TC) – Total number of lines in the routine including comments, declarations and the main body of the code.
- Number of code lines (CL) – Number of lines of executable code in the routine excluding the declaration and comment lines.
- Number of characters (Cr) – All characters in the routines.
- Number of comments (Cm) – For the Pascal routines, a comment is either a line beginning with test %%, or text in comment brackets, either of the form { < comment > } or (* < comment >*). For FORTRAN routines, a comment consists of the text on a line after either |, C or *.
- Number of comment characters (CC) – The amount of text found in the routines comments.
- Number of code characters (Co)– The amount of text which makes up the executable code in the routine.
- Halstead's Program Length (N'), where $N' = N_1' + N_2'$ and N_1' represents a total operator count and N_2' represents a total operand count [29]
- Halstead's Estimate of Program Length Metric (Ne), where Ne=$\eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$, and η_1 and η_2 represent the unique operator and operand counts, respectively[29].
- Jensen's Estimate of Program Length Metric (JE), where JE=$\log_2 \eta_1! + \log_2 \eta_2!$ [30].
- McCabe's Cyclomatic Complexity Metric (M), where $M = e - n + 2$, and e represents the number of edges in a control flow graph of n nodes [31].
- Belady's bandwidth metric (BW), where:

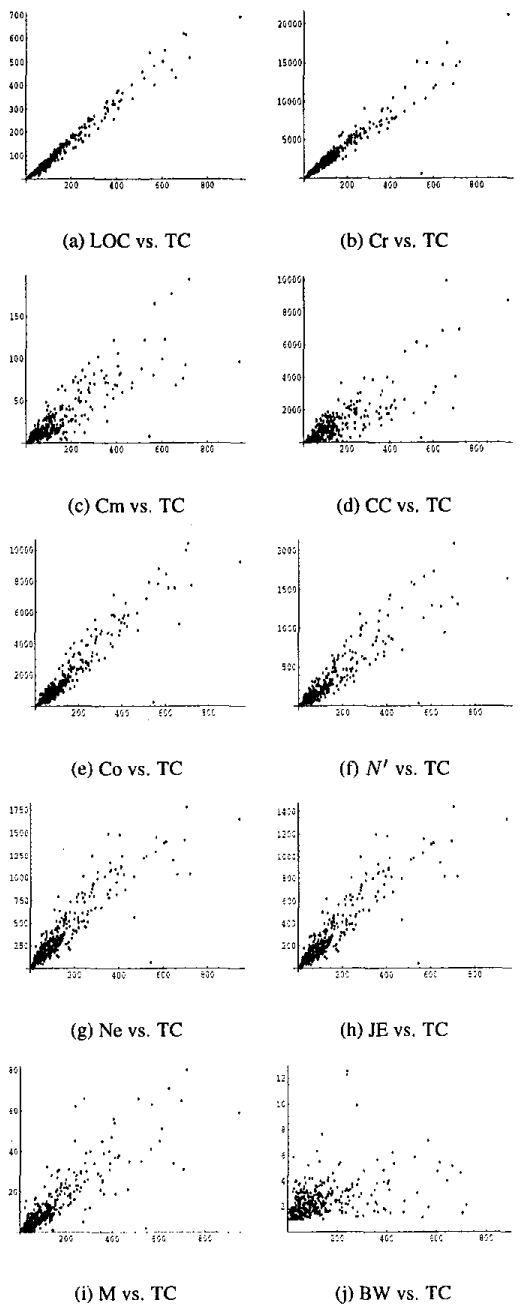
$$BW = \frac{1}{n}\sum_i iL_i \qquad (17)$$

and L_i represents the number of nodes at level i in a nested control flow graph of n nodes [30]. This metric indicates the average level of nesting or width of the control flow graph representation of the program.

By using these independent metrics as integrated complexity metrics, the random vector $\mathbf{x}$ is a 11-dimension vector with each metric as one component. Each vector $\mathbf{x}_i$ represents one sample point in the metric space, and we can apply the mixture model analysis in this high-dimension vector space to partition data samples into proper classes. When estimating mixture model parameters, we do not need to know the change requests (faults).

Principal Components Analysis(PCA): In a software development application, the independent variables (complexity metrics) may be strongly interrelated as they demonstrate a high degree of multicollinearity. We first examine the relationship of metric TC with other metrics, as shown in Figure 1.

It is clearly seen in Figure 1 that the metric TC has nearly linear relationship with some metrics such as LOC, Cr and Co. Several independent variables demonstrating a high degree of multicollinearity will have a negative effect on the regression model. One distinct result of multicollinearity in the independent variables is that the statistical models developed from them have highly unstable regression coefficients [7]. To reduce the interrelated effect, we adopt PCA (also called *Karhunen-Loéve transformation*) to transform the original complexity metrics space into an orthogonal



(a) LOC vs. TC (b) Cr vs. TC

(c) Cm vs. TC (d) CC vs. TC

(e) Co vs. TC (f) N' vs. TC

(g) Ne vs. TC (h) JE vs. TC

(i) M vs. TC (j) BW vs. TC

Figure 1. The relationship of metric TC with other metrics. From (a) to (j): horizontal axis is metric TC, vertical axes are metric LOC, Cr, Cm, CC, Co, N', Ne, JE, M and BW respectively. There are several metrics that exhibit multicollinearity.

vector space. The principle of PCA is simple. Let us assume the data set has a covariance matrix $\mathbf{\Sigma}$, which is a real symmetric matrix and can be decomposed as follows:

$$\mathbf{\Sigma} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T \tag{18}$$

where $\mathbf{U}$ is a matrix whose column i is the eigenvector $\mathbf{u}_i$, and $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues. Note that each of the eigenvectors is called a principal component. The vectors $\mathbf{x}$ are projected onto the eigenvectors to give the components of the transformed vectors $\mathbf{x}'$. That is,

$$\mathbf{x}' = \mathbf{U}^\mathbf{T}\mathbf{x}. \tag{19}$$

PCA can be used to reduce the dimension of the data space by taking $M < d$ eigenvectors corresponding to the first M largest eigenvalues to construct the transform matrix. The error introduced by a dimensionality reduction using PCA can be evaluated using

$$E_M = \frac{1}{2}\sum_{i=M+1}^{d} \lambda_i, \tag{20}$$

where the smallest $d - M$ eigenvalues λ_i and their corresponding eigenvectors are discarded.

The eigenvalues for the MIS data set are shown in the Table 1.

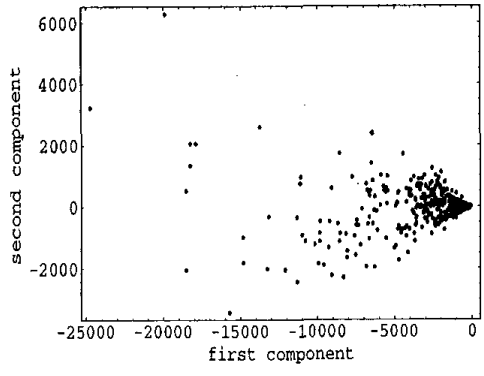
When using PCA to reduce the dimension of data space, we know from Table 1 that the first 7 components can represent main feature of the data set with a relatively small error (E_M =46.6338). However, some patterns are separable in high dimension space, but they become inseparable when projected into low dimension space. Therefore, we just apply PCA to transform data into an orthogonal set, using all 11-dimension in the data analysis. The results presented in this paper are based on PCA transformed data space, which is a 11-dimensional vector space. Figure 2 shows data distribution when projected onto first two principal components space and third-fourth principal components space.

For such a data space, each point represents one program module, which is characterized by its complexity metrics. These points can be assumed as samples arising from two or more models mixed in varying proportions. When the mixture model analysis with EM algorithm was applied to the 390 program modules in the PCA de-correlated 11-dimensional vector space, the most probable results are shown in Figure 3 for log likelihood function vs. model component number k as well as AIC vs. k.

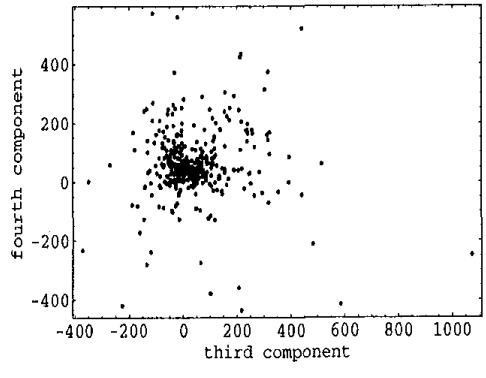
In Figure 3a, we can see that the log likelihood function of the system increases as the model number increases. Increasing model number makes finer classification for given software modules, and each model represents a subset of the data in which samples have similar characteristics. The

Table 1. The eigenvalues for the MIS data set

Component	1	2	3	4	5	6
Eigenvalue	1.28×10^7	6.05×10^5	1.71×10^4	1.34×10^4	4.77×10^3	2.41×10^3
Component	7	8	9	10	11	
Eigenvalue	1.78×10^2	47.2	31.5	13.5	0.98	

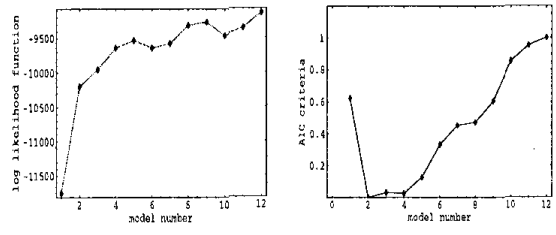


(a)



(b)

Figure 2. Data distribution in vector space (a) first two principal components and (b) third-fourth principal components.



(a) (b)

Figure 3. (a) The log likelihood function vs. model number. With the increase of the model number k, the function tends to increase too. (b) Typical results for AIC's vs. model number k for PCA de-correlated data set. The minima occurs at $k^* = 2$.

AIC model selection criterion in Figure 3b shows that with PCA de-correlated data set, classifying the modules into two groups is a proper selection. This gives us an insight into some intrinsic properties of the PCA de-correlated complexity metrics data set.

With two-class classification, the experimental results as obtained from Eq. (12) show that the module number in each group is $N_1 = 264$ and $N_2 = 126$, respectively. Note there are unequal sample numbers for the two-group classification.

The estimated mixture model parameters with EM algorithm for the case $k = 2$ are as the following:

Mixture weights: $\alpha_1 \approx 0.673$, and $\alpha_2 \approx 0.327$. Recall that $\alpha_j = \frac{1}{N}\sum_{i=1}^{N} p(j|\mathbf{x}_i)$, then $N_j = \sum_{i=1}^{N} p(j|\mathbf{x}_i) = \alpha_j N$. This should be the possible module number in class j. The obtained results are $N_1 \approx 0.673 \times 390 = 262$ and $N_2 \approx 0.327 \times 390 = 128$, respectively, which is agreeable with the experimental results obtained by using equation (12). As the mixture weights are a rough indication of module number distribution, this implies a high confidence in our results.

Mean vector: With two-class partition, the mean vector

for each group is shown in Table 2 for the original complexity metrics. The maximum and minimum values are also listed in Table 2 for reference. Notice that for the sake of readability, the values listed in Table 2 are transformed back from the PCA de-correlated space to the original data space.

The positions of the mean for each metric (i.e., m_1 and m_2) show the information to partition modules using single metric. Note that for all the 11 metrics, $m_2 > m_1$. This means class two consistently has a higher value than class one for all the metrics.

Covariance matrix: The covariance matrix is a symmetric matrix. Its diagonal element is the variance of each metric, while off-diagonal elements reflect the correlation between the metrics. (Refer to Eq.(7).) Here Table 2 only shows diagonal elements of the covariance matrices in the last two columns. Some metrics show high variance with two classes partition, implying that two-class partition is not the best choice from the point of view of minimal variance reduction.

The total module number is 390 in the given data set. With the two mixture models approach, the first group has 264 modules, while the second group has 126 modules, and the ratio is about 2/3 and 1/3 respectively. By the mixture model analysis, we now know that there are two classes for the given program modules: class one has more modules than class two for this data set. Furthermore, class two has higher complexity metrics values than class one.

Although at this stage we do not have failure data, we can pretty much determine that class one is non fault-prone while class two is fault-prone. The reason is two-fold. The first reason is that class two has consistently higher values of the complexity metrics, indicating its fault-prone nature. The second reason is that most (80%) of faults are found in a small portion (20%) of the software code, so we can label that the class with larger number of modules as non fault-prone class, and the class with less number modules as fault-prone class. Here we can see that very little prior knowledge about the number of faults is needed to develop this predictive model using mixture model with EM algorithm. This is the major advantage of our approach compared with previous model classification techniques published in the literature.

4 Quality Prediction Results and Discussion

4.1 Misclassification errors

The above analysis of program metrics with a mixture model can be obtained in early software develop stage. When the change of requests (CRs) become available later, we can use the CRs to assess the merit of the mixture model. The data analysis results are shown in Table 3.

There are two types of errors that can be made in the partition. A Type I error is the case where we conclude that a program module is fault-prone when in fact it is not. A Type II error is the case where we believe that a program module is non fault-prone when in fact it is fault-prone. Of the two types of errors, Type II error has more serious implications, since a product would be seem better than it actually is, and testing effort would not be directed where it would be needed the most.

When we consider module with 0 or 1 CRs to be non fault-prone, those with CRs from 18 to 98 to be fault-prone, then Type I error is 8.8% and Type II error is 12.8%. When modules with CRs from 10 to 98 are considered as fault-prone, then Type II error will rise to 28.1%. It is noted that in supervised learning such as feedforward neural network approach, the data set is partitioned into two parts: training samples and validation samples. The method of partition data set can have an effect on the prediction accuracy, as shown in the following experiment.

For MIS data set, there are 89 modules with CRs from 10 to 98, which are considered as fault-prone modules. Now let us randomly draw 30 modules (i.e., one third) from this subset of MIS data set. From mixture model analysis results, we can know the Type II error computed from these 30 modules. The Table 4 shows the experimental results of randomly drawing 30 samples from 89 modules without replacement, where the experiments are repeated 50 times. It can be known that the best result for Type II error is about 13%, which is the same as that of discriminant analysis method [7]. The statistical mean for Type II error is 27.1%, which is nearly the same as 28.1% obtained by the mixture model analysis based on all 89 modules.

4.2 Classification Probability

As stated in Section 2.3, assigning a module as either fault-prone or non fault-prone is based on Bayesian classification rule.

In two-model mixed case, the joint density of the system can be written in the form,

$$p(\mathbf{x}, \Theta) = \alpha_1 G(\mathbf{x}, \mathbf{m}_1, \mathbf{\Sigma}_1) + (1 - \alpha_1) G(\mathbf{x}, \mathbf{m}_2, \mathbf{\Sigma}_2). \tag{21}$$

The posterior probabilities become

$$p(1|\mathbf{x}) = \frac{\alpha_1 G(\mathbf{x}, \mathbf{m}_1, \widehat{\mathbf{\Sigma}}_1)}{p(\mathbf{x}, \Theta)}, \tag{22}$$

$$p(2|\mathbf{x}) = \frac{(1 - \alpha_1) G(\mathbf{x}, \mathbf{m}_2, \widehat{\mathbf{\Sigma}}_2)}{p(\mathbf{x}, \Theta)} = 1 - p(1|\mathbf{x}).$$

Figure 4 shows the two-component probability distribution of the joint density projected at each principal component axis. The solid line depicts the component

Table 2. Mean vector component as well as maximum and minimum value for each metric, and the diagonal values of covariance matrices obtained by ML with EM algorithm.

	min	max	m_1	m_2	Σ_1(diag.)	Σ_2(diag.)
TC	3	944	68.04	260.01	1565.7	26771
LOC	2	692	52.28	210.23	1125.9	18132
Cr	59	21266	1458	5620	766272	1.284×10^7
Cm	0	194	12.02	48.54	62.429	1258.87
CC	0	9946	561.52	1825	222703	2.703×10^6
Co	30	10394	761.37	3469	225432	4.573×10^6
$N\prime$	3	2083	137	629	7392.55	158213
Ne	2	1777.3	183.7	669.6	10534	135308
JE	0.8	1437.2	132.8	521.7	6143.7	89333
M	1	80	5.76	24.56	12.507	249.496
BW	1	12.56	2.1	3.13	0.78547	3.774

Table 3. The classification for MIS data set by mixture model analysis.

CRs	0,1	2,3	4,5	6,7	8,9	10,11	12,13	14,15	16,17	18-98
Number of group 1	104	66	33	25	11	9	6	1	4	5
Total modules	114	78	49	36	24	19	12	10	9	39
Percent of group 1	91.2	84.6	67.3	69.4	45.8	47.4	50	10	44	12.8

$\alpha_1 G(\mathbf{x}, \mathbf{m}_1, \mathbf{\Sigma}_1)$, while the dashed line depicts the component $(1-\alpha_1)G(\mathbf{x}, \mathbf{m}_2, \mathbf{\Sigma}_2)$. At each point, the value of each probability component is proportional to the value of the posterior probability. When we use Bayesian decision to classify program module i into class j, the misclassification risk can be obtained with Figure 4. If the position of a module is at or near the position at which the values of the two components are nearly equal, (i.e., where the solid line and the dashed line intersect in each figure) the misclassification risk will be high.

Each principal component metric is a linear combination of the original complexity metrics. When we predict that one program module is possible of either fault-prone or non fault-prone, the decision is made by combining all principal components together, not just a single metric. Combining all metrics to predict the software quality is one of the way to reduce the risk of misclassification.

4.3 Advantages of Mixture Model Analysis

Building model to support the prediction of software quality based on software complexity metrics can be quite challenging due to various inherent constraints. Sometimes the values of complexity metrics are not complete because it needs a long time collecting them, and building models requires the use of complete data types of variables. The EM algorithm was originally developed for incomplete data set, therefore the approach described above can handle the types of variables with partial missing values. Other methods such as regression tree modeling [32], feedforward neural networks [12] requires to know the target value (fault number) in advance, and regression tree modeling also needs to assign a threshold to split the data set. On the other hand, in the mixture model analysis with EM algorithm, only little prior knowledge is needed to predict the module characteristics based on the complexity metrics.

The mixture model analysis method also does not require an equal class number, so it is a more general model and classification rule used than that discriminant analysis [7]. In the linear discriminant analysis, the covariance matrices are assumed the same for all classes, which is seldom the case in the real world.

Furthermore, if we suppose that the mixture model classification result is correct, from the results shown in Table 3, we know that the most non fault-prone modules should have no more than 3 CRs, which has the percentage greater than 88%. Furthermore, the modules with CRs from 4 to 17 should be mediately fault-prone modules, and the modules with CRs 18 to 98 is the fault-prone group. This shows that the mixture model can help us gain an insight in the relationships between the software complexity metrics and the number of faults in the module.

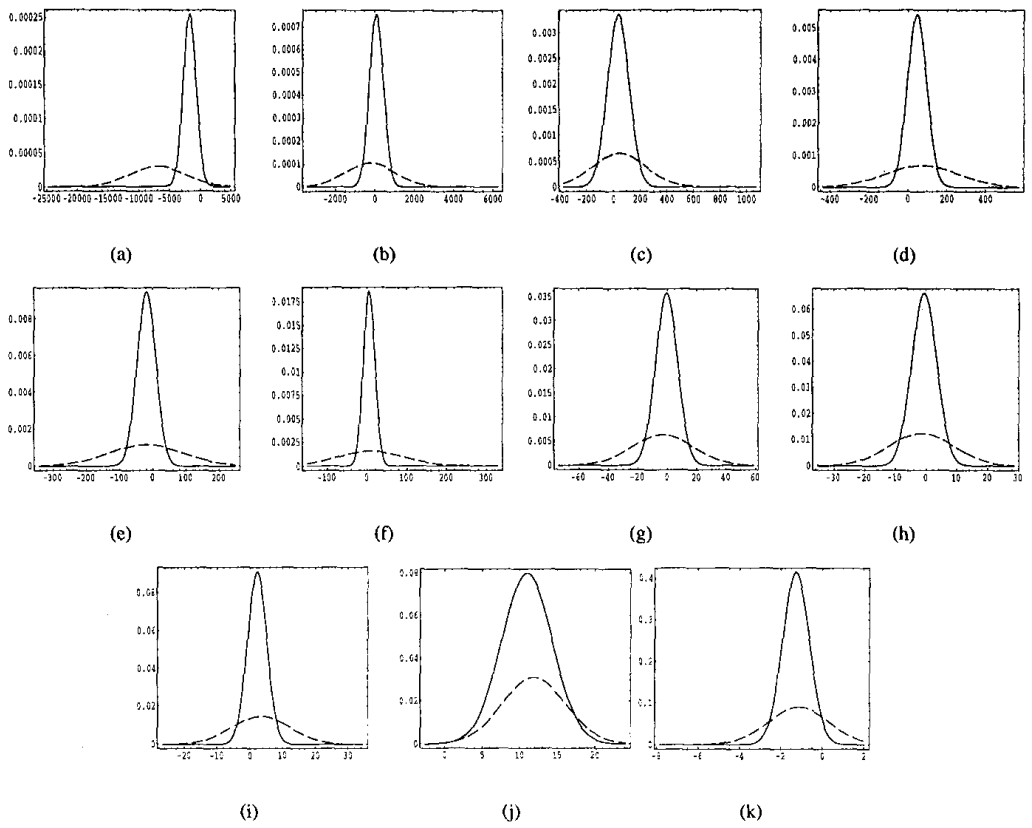


Figure 4. The plot for two components of the joint density projected at principal axis, the figures from (a) to (k) is corresponding to the 11 principal component axes in order.

Table 4. Misclassification rate for randomly drawing 30 samples out of 89 modules without replacement. The mean and standard deviation are computed based on 50 times repeated experiments.

	min.	max.	mean	std.
misclass. rate	0.133	0.40	0.271	0.064

5 Conclusion

Software metrics can reveal a lot of information about the code at several stages of development. They can identify the routines which need to be redesigned due to higher complexity, routines which may require thorough testing, and features which may require more support. The mixture model with EM algorithm is a novel way to analyze software metrics, to understand the involved relationships among them, to identify the fault-prone modules, and thus to take remedial actions before it is too late. Based on the experimental results, this modeling approach provides an effective way to predict software quality in a very early stage of program development.

References

[1] M. R. Lyu, *Handbook of software Reliability Engineering,*

IEEE Computer Society Press, McGraw Hill, 1996.

[2] B. W. Boehm and P. N. Papaccio, "Understanding and controlling software costs," *IEEE Trans. on Software Engineering*, vol. 14, no. 10, pp. 1462–1477, October 1988.

[3] F. G. Sayward A. J. Perlis and M. Shaw, *Software Metrics: An Analysis and Evaluation*, MIT Press, Cambridge, MA, 1981.

[4] V. Y. Shen, T.Yu, S. M. Thebaut, and L. R. Paulsen, "Identifying error-prone software—an empirical study," *IEEE Trans. on Software Engineering*, vol. SE-11, pp. 317–323, April 1985.

[5] S. G. Crawford, A. A. McIntosh, and D. Pregibon, "An analysis of static metrics and faults in C software," *J. Syst. Sofyware*, vol. 5, pp. 27–48, 1985.

[6] L. C. Briand, V. R. basili, and C. Hetmanski, "Developing interpretable models for optimized set reduction for identifying high-risk software components," *IEEE Trans. on Software Engineering*, vol. SE-19, no. 11, pp. 1028–1034, November 1993.

[7] J. Munson and T. Khoshgoftaar, "The detection of fault-prone programs," *IEEE Trans. on Software Engineering*, vol. SE-18, no. 5, pp. 423–433, May 1992.

[8] T. Khoshgoftaar and J. Munson, "Predicting software development error using software complexity metrics," *IEEE Trans. on Software Engineering*, vol. 8, no. 2, pp. 253–261, February 1990.

[9] A. A. Porter and R. W. Selby, "Empirically guided software development using metric-based classification trees," *IEEE Software*, vol. 7, no. 2, pp. 46–54, March 1990.

[10] R. W. Selby and A. A. Porter, "Learning from examples: Generation and evolution of decision trees for software resource analysis," *IEEE Trans. on Software Engineering*, vol. 14, no. 12, pp. 1743–1756, December 1988.

[11] L. C. Briand, V. R. basili, and W. M. Thomas, "A pattern recognition approach for software engineering data analysis," *IEEE Trans. on Software Engineering*, vol. SE-18, no. 11, pp. 931–942, November 1992.

[12] D. L. lanning T. Khoshgoftaar and A. S. Pandya, "A comparative study of pattern recognition techniques for quality evaluation of telecommunications software," *IEEE J. Selected Areas in Communication*, vol. 12, no. 2, pp. 279–291, February 1994.

[13] L. M. Ottenstein, "Quantitative estimates of debugging requirements," *IEEE Trans. on Software Engineering*, vol. SE-5, no. 2, pp. 504–514, September 1979.

[14] C.M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, 1995.

[15] N. M. Laird A. P. Dempster and D. B. Rubin, "Maximum-likelihood from incomplete data via the EM algorithm," *J. Royal Statist. Society*, vol. B39, pp. 1–38, 1977.

[16] R. A. Redner and H. F. Walker, "Mixture densities, maximum likelihood and the em algorithm," *SIAM Review*, vol. 26, pp. 195–239, 1984.

[17] K. E. Basford G. J. McLachlan, *Mixture Models:Inference and applications to clustering*, Dekker, New York, 1988.

[18] D. Hand B. S. Everitt, *Finite mixture distributions*, Chapman and Hall, London, 1981.

[19] N. E. Day, "Estimating the component of a mixture of normal distributions," *Biometrika*, vol. 56, pp. 463–474, 1969.

[20] H. H. Bock, "Probability models and hypotheses testing in partitioning cluster analysis," in *clustering and classification*, Riverside, California, 1996, pp. 377–453, World Scientific Press.

[21] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. AC-19, pp. 716–723, 1974.

[22] H. Bozdogan, *Multiple Sample Cluster Analysis and Approaches to Validity Studies in Clustering Individuals*, Doctoral dissertation, University of Illinois ar Chicago Circle, Chicago,IL, 1981.

[23] H. Bozdogan, "Modle selection and Akaike's information criterion: The general theory and its analytical extensions," *Psychometrika*, vol. 52, no. 3, pp. 345–370, 1987.

[24] G. Schwarz, "Estimating the dimension of a model," *The annals of Statisticals*, vol. 6, no. 2, pp. 461–464, 1978.

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

[26] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, Boston, second edition, 1990.

[27] V. R. Basili and D. H. Hutchens, "An empirical study of a syntactic complexity family," *IEEE Trans. on Software Engineering*, vol. SE-9, no. 6, pp. 664–672, November 1983.

[28] R. K. Lind, "An experimental study of software metrics and their relationship to software error," M.S. thesis, University of Wisconsin-Milwaukee, Milwaukee, December 1986, Master's thesis.

[29] M. Halstead, *Elements of Software Science*, New York Elsevier, North-Holland, 1977.

[30] H. Jensen and K. Vairavan, "An experimental study of software metrics for real-time software," *IEEE Trans. on Software Engineering*, vol. SE-11, no. 2, pp. 231–234, February 1994.

[31] T. J. McCabe, "A complexity measure," *IEEE Trans. on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.

[32] S. S. Gokhale and M. R. Lyu, "Regression tree modeling for the prediction of software quality," in *Proceedings of Third ISSAT International Conference:Reliability & Quality in Design*, Hoang Pham, Ed., Anaheim, CA, 1997, pp. 31–36.