IMPLEMENTATION OF A THREE-GROUP CLASSIFICATION MODEL USING CASE-BASED REASONING

by

Huiming Song

A Thesis Submitted to the Faculty of

The College of Engineering

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering

Florida Atlantic University

Boca Raton, Florida

August 2001

IMPLEMENTATION OF A THREE-GROUP CLASSIFICATION MODEL USING CASE-BASED REASONING

by

Huiming Song

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Taghi M. Khoshgoftaar, Department of Computer Science and Engineering, and has been approved by the members of his supervisory committee. It was submitted to the faculty of The College of Engineering and was accepted in partial fulfillment of the requirements for the degree of Master of Engineering.

SUPERVISORY COMMITTEE:

_____

Thesis Advisor

_____

_____

_____

_____

Chairman, Department of
Computer Science and Engineering

_____

Dean, College of Engineering

_____ _____

Vice Provost Date

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Taghi M. Khoshgoftaar, who spent lots of time on advising me in my research work. I sincerely appreciate him for giving me the opportunity to work with him. His inspiration, profound knowledge, wise thoughts and kind guidance were of great help to complete this thesis. I want to thank Dr. Eduardo Fernandez and Dr. Martin K. Solomon for agreeing to serve on my thesis committee and reviewing my thesis.

Special thanks to Dr. Robert Szabo. His previous research made the work possible. I used the same data set that he used in his work.

I deeply appreciate Zhiwei Xu, Kehan Gao and Naeem Seliya in helping me use LaTeXand reviewing my thesis. Special thanks to Kehan Gao for her work to verify some algorithms implemented in SMART by using an independent program.

I am very grateful to my wife, Yi Liu, and my parents in China for providing me with the strongest moral support.

ABSTRACT

Author:	Huiming Song
Title:	Implementation of A Three-Group Classification Model Using Case-Based Reasoning
Institution:	Florida Atlantic University
Thesis Advisor:	Dr. Taghi M. Khoshgoftaar
Degree:	Master of Engineering
Year:	2001

Reliability is becoming a very important and competitive factor for software products. Software quality models based on software metrics provide a systematic and scientific way to detect software faults early and to improve software reliability. Classification models for software quality usually classify observations using two groups. This thesis presents a new algorithm for classification using three groups, i.e., Three-Group Classification Model using Case Based Reasoning. The basic idea behind the algorithm is that it uses the commonly used two-group classification method three times. This algorithm can be implemented with other techniques such as logistic regression, classification tree models, etc. This work compares its quality with the Discriminant Analysis method. We find that our new method performs much better than Discriminant Analysis. We also show that the addition of object-oriented software measures yielded a model that a practitioner may actually prefer over the simpler procedural measures model.

To my dear wife and my dear daughter

CONTENTS

TABLES . . . . . **vii**
FIGURES . . . . . **ix**

1 MODELING METHODOLOGY **1**

1.1 Case-Base Reasoning 1
1.2 CBR Classification 3
1.3 Similarity Functions 4
1.4 Data Clustering 6
1.5 Classification with Three Groups 8
1.6 Statistical Modeling Methodology 14
1.7 Discriminant Modeling 15
1.8 Stepwise Model Selection 22
1.9 Evaluation of Models 23

BIBLIOGRAPHY **25**

TABLES

1.1 Summary of Two Group Model Error Types 11

1.2 Summary of Three Group Model Error Types 11

1.3 Summary of Two Group Model Error Types 17

1.4 Summary of Three Group Model Error Types 17

FIGURES

1.1 The Figure of CBR Classification with Three Groups . . . 10

1.2 The General Interface of CBR Classification with Three Groups 12

1.3 The Interface of CBR Classification with Three Groups . 13

Chapter 1

MODELING METHODOLOGY

In this chapter, we introduce the methods of Case-Based Reasoning and discriminant analysis which were used in our case study. It also gives an overview of principle components analysis and Z-test.

1.1 Case-Base Reasoning

Case-Based Reasoning (CBR) is a modeling technique that seeks to answer new questions by identifying similar "cases" from the past. CBR is a part of the computational intelligence field focusing on automating reasoning processes. When applied to software reliability, the working hypothesis of our approach is: a module currently under development is probably fault-prone, if a module with similar product and process attributes in an earlier release was fault-prone. It finds solutions to new problems based on past experience, represented by "cases" in a "case library". Each module is a "case" of software development. The case library and the associated retrieval and decision rules constitute a CBR model. Fault-prone was defined by a threshold on the number of faults discovered during integration and testing.

In our study, we focus on classification problems in software quality modeling. Suppose each case in the library has known attributes and class membership. Given a case with unknown class, we predict its class to be the same as the class of the most similar case in the library, where similarity is defined in terms of case attributes. Our study applies this approach to classification of software modules as Green, Yellow, and Red, according to the risk of fault prone.

In principle, CBR has several advantages over statistical classification techniques.

- CBR can be designed to alert users when a new case is outside the bounds of current experience.
- Once the most similar case has been selected from the library, its detailed description can help one interpret the automated classification.
- CBR systems can add new cases to the library and remove obsolete cases from the library. Thus, a case-based reasoning can take advantage of new or revised information as it becomes available.
- CBR is scalable to very large case libraries, and is amenable to concurrent retrieval techniques.

1.2 CBR Classification

A case consists of all available information on a module. This includes its Class and all measurements. Supplementary information on the module's development history may also be included for human interpretation. In our study, the case library consists of the fit data set, having n modules. The case library represents the past experiences of the development organization. Let $\mathbf{c_k}$ be the vector of attributes of the k^{th} module in the case library, and let c_{kj} be the j^{th} attribute of that module.

From the case library we want to extract the one that most closely resembles our unclassified module. Let $\mathbf{x_i}$ be the vector of attributes of the i^{th} unclassified module, and x_{ij} be the j^{th}attribute of that module.

Since our data is strictly quantitative, we think of similarity as the distance between cases. Consider each attribute to be a dimension of a multi-dimensional space. Each case is represented by a point in this space, and the unclassified module by another point. We think of "similarity" as the inverse of the "distance" between modules. We provide several similarity functions to calculate a distance, d_{ij}, between the target module $\mathbf{x_i}$ and every case $\mathbf{c_j}$ in the case library.

1.3 Similarity Functions

Several measures of similarity are presented in the literature according to the problem domain, the availability of attribute data, and whether data types are categorical, discrete, real, etc. For quantitative attributes, the Euclidean distance and city-block distance are commonly used. The Mahalonobis distance has the advantage of explicitly accounting for correlations among attributes. It should be noted that the raw metrics collected, i.e., the independent variables, are often measured in varying ways and could contain a wide variety of ranges and scales. With this being the case, it is often beneficial to first standardize the metrics. Standardization is a method that allows all metrics to use the same unit of measure. For each metric, X_i, the standardized metric is computed according to the following formula:

$$Z_i = \frac{X_i - \overline{X_i}}{S_i} \tag{1.1}$$

$\overline{X_i}$ is the mean, and S_i is the standard deviation of the i^{th} metric X_i. Standardization is not necessary for all of the similarity functions available. In fact, the Mahalonobis Distance similarity function does not require that standardization be used. In our study, we support the following similarity functions.

Absolute Difference Distance: This distance is also known as City-Block Distance or Manhattan Distance. It is calculated by taking the weighted sum of the absolute value of the difference in independent variables between the current case and a past case. The user of the model provides the weights, and the absolute

value is taken because the direction of the difference in distance is irrelevant. This distance is primarily used for quantitative attributes. The following is the equation for Absolute Difference Distance:

$$d_{ij} = \sum_{k=1}^{m} w_k |c_{jk} - x_{ik}| \tag{1.2}$$

where m is the number of independent variables, and w_k is the weight of the k^{th} independent variable.

Euclidean Distance: This distance views the independent variables as dimensions within an m-dimensional space, with m being the number of independent variables. A current case is represented as a point within this space. The distance is calculated by taking the weighted distance between a current case and a past case within this space. Again, the user of the model provides the weights, and this distance is also commonly used when the data set contains quantitative attributes. The following is the equation for Euclidean Distance:

$$d_{ij} = \sqrt{\sum_{k=1}^{m} w_k (c_{jk} - x_{ik})^2} \tag{1.3}$$

Mahalonobis Distance: This distance measure is an alternative to the Euclidean distance. It is used when the independent variables are highly correlated. Mahalonobis Distance is a very attractive similarity function to implement because:

- It can explicitly account for the correlation among the attributes.

- The independent variables do not need to be standardized.

In cases where the variances of the independent variables have unit variances and are uncorrelated, the Mahalonobis distance is simply the square of the Euclidean Distance. The following is the equation for Mahalonobis distance:

$$d_{ij} = (\mathbf{c_j} - \mathbf{x_i})' S^{-1} (\mathbf{c_j} - \mathbf{x_i}) \tag{1.4}$$

Prime (') means transpose, and S is the variance-covariance matrix of the independent variables over the entire case library. S^{-1} is its inverse.

1.4 Data Clustering

There are several classification methods that can be used to enhance the classification of the dependent variable(s) in the software quality classification models.

Data clustering is such a method that can be used in CBR as an enhancement to the classification models. The case library is partitioned into clusters according to the actual class of each case. The average distances to the clusters are then computed for the current case. The current case's classification is determined by comparing the ratio of these average distances to the cost ratio.

Cost ratio is defined as C_I/C_{II}, where C_I is the cost of a Type I misclassification or error, and C_{II} is the cost of a Type II misclassification or error. A Type I misclassification is when a model classifies a case as fault-prone when it is actually not fault-prone. A Type II misclassification is when a model classifies a case as not

fault-prone when it is actually fault-prone [2]. The Type II misclassifications are generally considered more serious than the Type I misclassifications because they represent the cost of releasing fault-prone cases into production and fixing fault-prone cases after they have been released and deployed. Type I misclassifications, on the other hand, represent the wasted efforts on analyzing low-risk modules. In our study, we extend the usage of Type I and Type II misclassification. They are not limited to the distinguish the misclassification of fault-prone and not fault-prone, but to distinguish the errors from one group to another.

During the model development, the user can experiment with various cost ratios to reach their desired value for the model. Because organizations may have a preference as to which type of error they would like to minimize, we did not assume a specific error type to be more important. Instead, we chose the cost ratio that would provide the most balanced Type I and Type II misclassification rates. The classification terminology for data clustering is as follows [1]: for an unclassified case, x_i, let $d_{nfp}(x_i)$ be the average distance to the not fault-prone nearest neighbor cases, and let $d_{fp}(x_i)$ be the average distance to the fault-prone nearest neighbor cases.

The following is the classification rule for data clustering analysis:

$$Class(x_i) = \begin{cases} \textit{not fault-prone} & \text{if } \frac{d_{fp}(x_i)}{d_{nfp}(x_i)} > \frac{C_{II}}{C_I} \\ \textit{fault-prone} & \text{otherwise} \end{cases} \tag{1.5}$$

For multi-classification, there are more than two misclassifications. But we can easily rewrite the classification rules according to the basic idea. We listed the corresponding rules in next section. In our case study, the cost-ratio ranges from 0.1 to 1.0 with a step of 0.05. This can be easily changed using our SMART (The Software Measurement Analysis and Reliability Toolkit) [13].

1.5 Classification with Three Groups

Dr. Khoshgoftaar[7, 8] developed a new algorithm based on the above. Here we call the new algorithm as Three-Group Classification Model Using Case-Based Reasoning. This new algorithm uses the traditional method of two groups three times. First according to the number of faults, he partitioned the case library into three groups, Green, Yellow and Red. The following four steps described this new algorithm.

Step1: Compute the distance of each module of the unclassified cases with all cases in the Green and Red groups using a similarity function. According to the distances, we parse the unclassified cases into two parts, named Test-I and Test-II. The classification rule is as following:

$$Class(x_i) = \begin{cases} Green & \text{if } \frac{d_{Red}}{d_{Green}} > \frac{C_{RG}}{C_{GR}} \\ Red & \text{otherwise} \end{cases} \tag{1.6}$$

where d_{Green} and d_{Red} represent the average distance to the group of Green and Red. C_{GR} is the cost of Type_GR error, C_{RG} is the cost of Type_RG error.

Step2: Compute the distance of each modules of the Test-I data set with all cases in the Green and Yellow groups using the same similarity function as **Step1**. According to the distances, we parse the Test-I data set into two parts, the Test-Green group and the Test-Yellow1 group. The classification rule is as following:

$$Class(x_i) = \begin{cases} Green & \text{if } \frac{d_{Yellow}}{d_{Green}} > \frac{C_{YG}}{C_{GY}} \\ Yellow & \text{otherwise} \end{cases} \tag{1.7}$$

where d_{Green} and d_{Yellow} represent the average distance to the group of Green and Yellow. C_{GY} is the cost of Type_YG error, C_{GY} is the cost of Type_GY error.

Step3: Compute the distance of each modules of the Test-II data set with all cases in the Yellow and Red groups using the same similarity function. According to the distances, we parse the Test-II data set into two parts, Test-Red group and Test-Yellow2 group. The classification rule is as following:

$$Class(x_i) = \begin{cases} Yellow & \text{if } \frac{d_{Red}}{d_{Yellow}} > \frac{C_{RY}}{C_{YR}} \\ Red & \text{otherwise} \end{cases} \tag{1.8}$$

where d_{Yellow} and d_{Red} represent the average distance to the group of Yellow and Red. C_{RY} is the cost of Type_RY error, C_{YR} is the cost of Type_YR error.

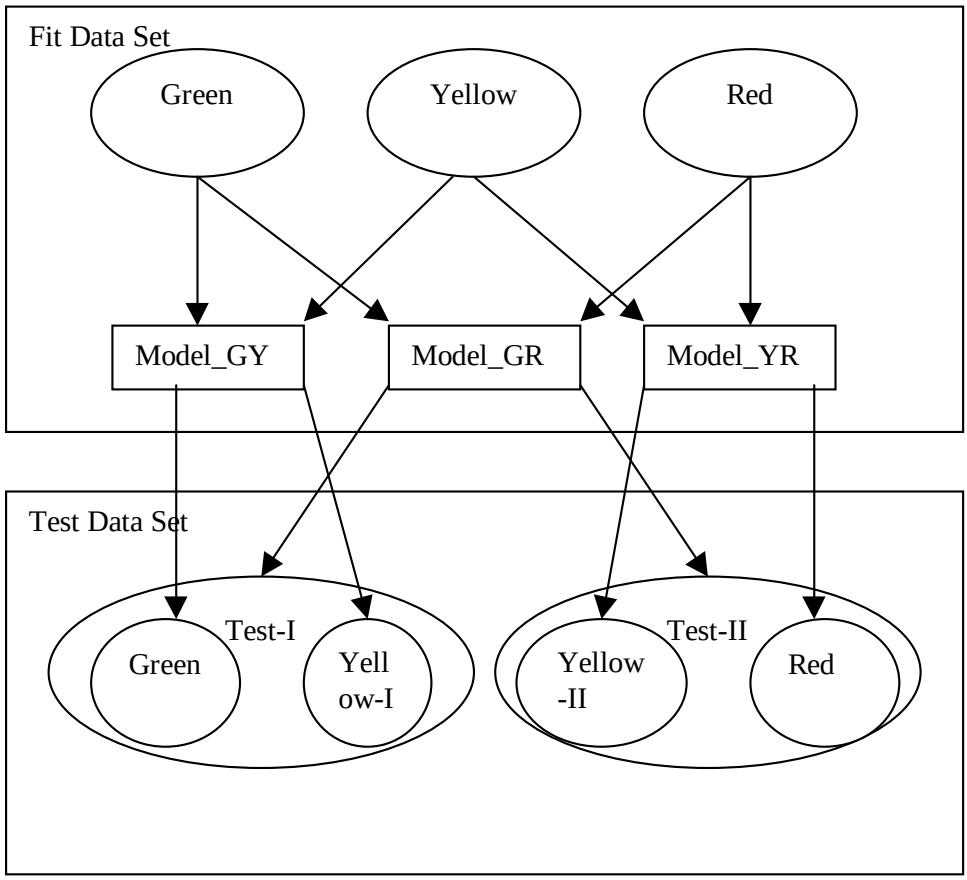

Figure 1.1: The Figure of CBR Classification with Three Groups

Step4: At last, we get Test-Yellow group by combining Test-Yellow1 group and Test-Yellow2 group. Thus we have classified the test data set into three groups: Test-Green, Test-Yellow and Test-Red.

Figure 1.1 shows the idea of this algorithm.

These three groups can lead to six misclassifications.

- Type_GR: A Green module is classified as Red.

Table 1.1: Summary of Two Group Model Error Types

Actual Risk Group	**Predicted Risk Group**	
	High	Low
Low	**Type I**	–
High	–	**Type II**

Table 1.2: Summary of Three Group Model Error Types

Actual Risk Group	**Predicted Risk Group**		
	High	Medium	Low
Low	**Type 1**	**Type 2**	–
Medium	**Type 3**	–	**Type 4**
High	–	**Type 5**	**Type 6**

- Type_GY: A Green module is classified as Yellow.
- Type_YR: A Yellow module is classified as Red.
- Type_YG: A Yellow module is classified as Green.
- Type_RY: A Red module is classified as Yellow.
- Type_RG: A Red module is classified as Green.

Table **??** summarizes the misclassifications. This new algorithm has been implemented using C++ and added as a new feature to SMART version 2.0. Figure 1.2 and Figure 1.3 show the user GUI general and tab page interfaces in SMART.

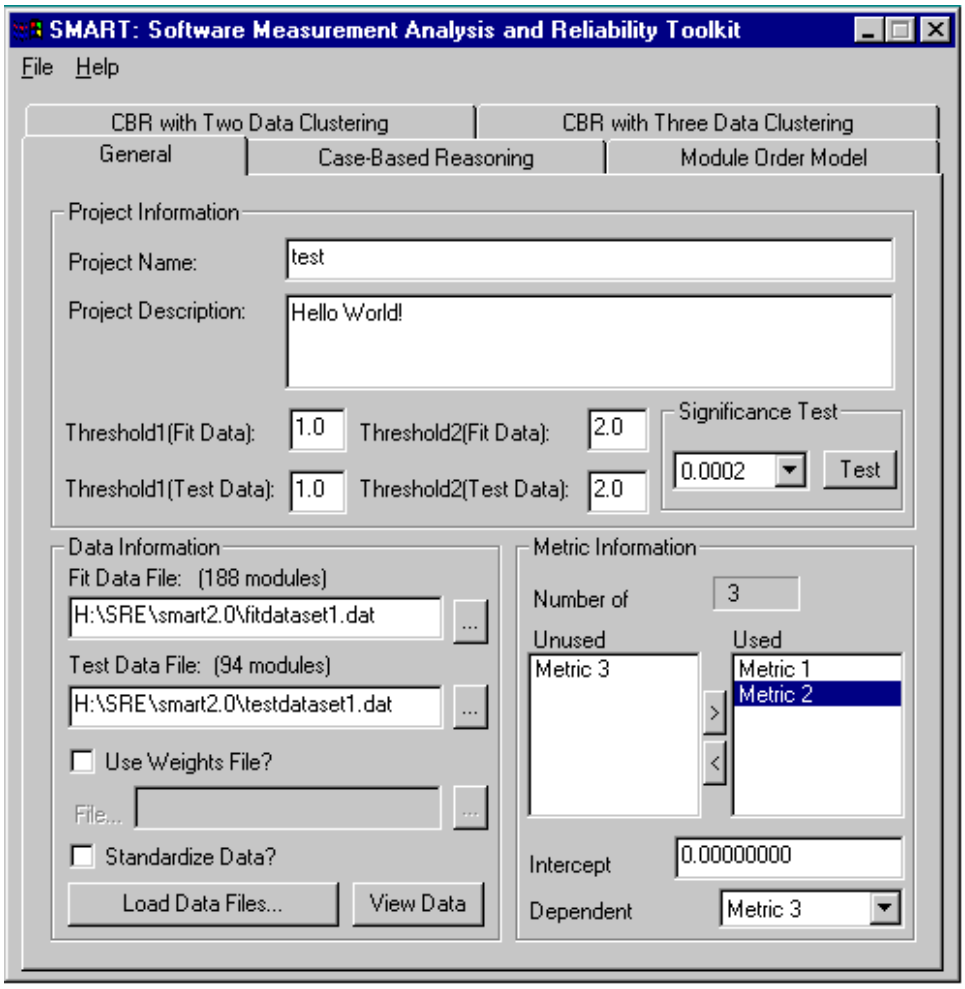

Figure 1.2: The General Interface of CBR Classification with Three Groups

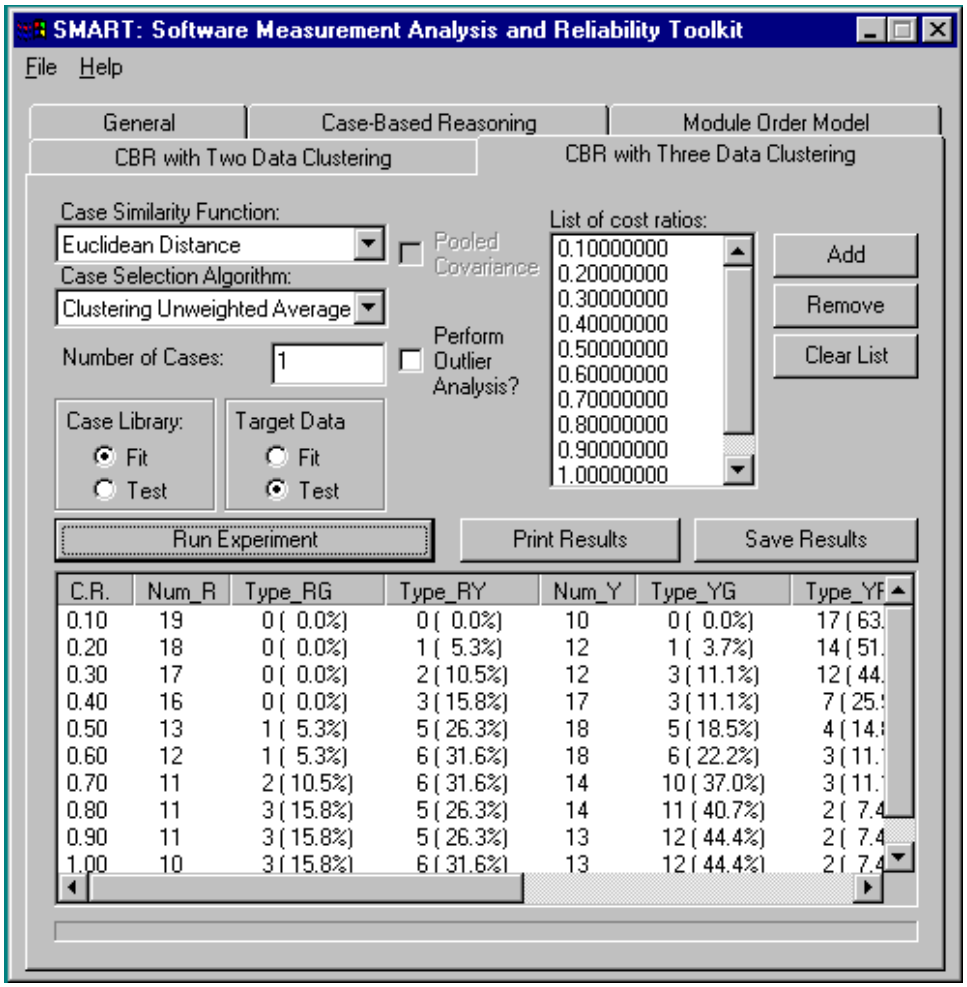

Figure 1.3: The Interface of CBR Classification with Three Groups

1.6 Statistical Modeling Methodology

In this section, we discuss the methodology of Discriminant Analysis. We will compare this method with Case-Based Reasoning Classification.

The measurement of software is becoming an established discipline in many software organizations. Typically, we attempt to predict the quality of a software system based on some quantifiable measures. These measures can then be used as input to a predictive or classification model. Using these models, the development process can be modified to help improve the quality of the system in a manner that is repeatable across similar environments.

The set of available software metrics is quite large. For example, program size could be expressed as lines of code, the number of executable statements, and many of the Software Science metrics proposed by Halstead [5]. Furthermore, these metrics often tend to be highly correlated. This correlation of the metrics is called *multicollinearity* .

To circumvent the problem of multicollinearity, researchers often limit their study to a few carefully selected software measures. On the other hand, taking as many metrics as possible into consideration should lead to a more complete model, since each measurement assesses a particular but sometimes overlapping aspect of the software. Munson and Khoshgoftaar addressed this issue by proposing the application of principal components analysis when developing a multiple linear regression model [18]. Furthermore, it has been shown that regression models and

neural network models using principal components can perform better than the corresponding model using the raw data [10, 11]. In Section 1.7, we will discuss discriminant modeling. For a discussion about principal components analysis, refer to [3].

1.7 Discriminant Modeling

Discriminant analysis is a statistical method that determines the optimum assignment of observations to two or more distinct groups based upon one or more quantitative measurements. These measurements are assumed to differ from group to group. Given a set of observations with known group memberships, i.e., a *fitting data set*, the methodology develops an assignment rule such that the chance of misclassification is minimized. The resulting model may be used to classify future observations based on the observed quantitative measures.

Discriminant analysis has been demonstrated to be useful for classifying high and low risk program modules [21, 9]. In this context, risk refers to the number of faults likely to be in the program module under consideration. The point at which the high and low risk boundary is drawn is subjective, and will vary from environment to environment. For example, modules with more than ten faults can be considered high risk while those with less than ten faults are considered low risk.

In order to distinguish from the terms we used before like Green, Yellow and Red, we apply discriminant analysis to build a model that classifies program

modules as high, medium, and low risk. Essentially there is no difference between Green, Yellow, Red and low, medium, high risk group. The observations used to fit the discriminant model are based on program files. The quantitative measurements upon which classification is based (independent variables) are principal components derived from a set of software product measurements extracted directly from the source code. Many common software product measures tend to be correlated while principal components are orthogonal [10]. Thus, principal components are favored as we can avoid the model selection problems that stem from correlations among the independent variables [3]. Though the large set of correlated measures will likely provide more information, the first few principal components will capture a large proportion of the software measurement variance.

The classification (dependent) variable, $Faults$, is a measure of the number of errors that will be detected at the end of a specific development phase. The modules are divided into three groups based on two cutoff values. Modules exceeding the high cutoff point are assigned to the high risk group or Red. Those less than or equal to the low cutoff are assigned to the low risk group or Green. Finally, modules greater than the low cutoff and less than or equal to the high cutoff are assigned to the medium risk group or Yellow. The cutoff values clearly determine the size of each group and will vary from environment to environment. Typically, the cutoff values are determined based on the past history of projects developed in a similar environment.

Table 1.3: Summary of Two Group Model Error Types

Actual Risk Group	**Predicted Risk Group**	
	High	Low
Low	**Type I**	–
High	–	**Type II**

Table 1.4: Summary of Three Group Model Error Types

Actual Risk Group	**Predicted Risk Group**		
	High	Medium	Low
Low	**Type 1**	**Type 2**	–
Medium	**Type 3**	–	**Type 4**
High	–	**Type 5**	**Type 6**

This way can lead to 6 type of misclassifications.

- Type 1: A low risk module is classified as high risk.
- Type 2: A low risk module is classified as medium risk.
- Type 3: A medium risk module is classified as high risk.
- Type 4: A medium risk module is classified as low risk.
- Type 5: A high risk module is classified as medium risk.
- Type 6: A high risk module is classified as low risk.

Table 1.4 summarizes the error type hierarchy for this three group classification model.

Of the six error types listed, Type 1 and Type 6 errors are considered the most serious. Each represents the maximum misclassification possible as measured by the distance between the actual and predicted groups. A Type 1 misclassification error occurs when a low risk module is classified as high risk. The consequences of such an error type are similar to the Type I case for the two group model. A Type 6 misclassification happens when a high risk module is classified as low risk. This error type has consequences similar to the Type II case for the two group model. This suggests that the cost of a Type 6 error is higher than a Type 1 error. A good model will minimize the number of Type 6 errors.

The three group discriminant model, will place a module into one of three groups based upon its probability of membership. For example, if the probability of membership into the three groups are 0.1, 0.3, and 0.6 respectively, the module will be placed into the group corresponding to the membership probability of 0.6. Remember that the probability of membership is a function of the independent variables. In our case, they are the principal components derived from the observed software product metrics. For some modules that are correctly classified, the membership probability will be much greater than the sum of the membership probabilities of the remaining groups. This indicates a high probability of correct classification. Conversely, some modules correctly classified will have probabilities less than the sum of the membership probabilities of the remaining groups indicating a lower probability of correct assignment. For those modules correctly classified

into group G_i, $1 - q_i$ is a measure of the *uncertainty* of the classification, where q_i is the probability of membership to G_i.

Before we can discuss discriminant analysis in more detail, we need to introduce the following notation:

- $\mathbf{x}$ is a p-dimensional vector containing the quantitative variables comprising an observation,
- g is the number of groups,
- G_i are mutually exclusive groups,
- n_i is the number of observations in group i of the fitting set,
- $f_i(\mathbf{x})$ is the probability of $\mathbf{x} \in G_i$,
- π_i is the prior probability of membership, or proportion of observations in G_i, and
- $q_i(\mathbf{x})$ is the posterior probability of membership in G_i given $\mathbf{x}$.

In general, an observation $\mathbf{x}_{ij}$, where $i = 1, 2, \ldots, g$, and $1 \leq j \leq n_i$, is a vector of p principal components derived from software product measures for the j^{th} program module of group G_i. For each group G_i, a certain proportion of modules, π_i, will fall into that group. For $i = 1, 2, \ldots, g$, $f_i(\mathbf{x})$ is the probability density

function of those $\mathbf{x}$ falling into group G_i. To minimize the total probability of misclassification, the model assigns the observation $\mathbf{x}$ to group G_i if

$$\pi_i f_i(\mathbf{x}) \geq \pi_j f_j(\mathbf{x}), \quad j = 1, 2, \ldots, g. \tag{1.9}$$

The assignment on the boundary is arbitrary. For example when $g = 2$, the model assigns the observation $\mathbf{x}$ to group G_1 if

$$\frac{f_1(\mathbf{x})}{f_2(\mathbf{x})} > \frac{\pi_2}{\pi_1}, \tag{1.10}$$

and to group G_2 otherwise. When $f_1(\mathbf{x})/f_2(\mathbf{x}) = \pi_2/\pi_1$, the assignment is arbitrary.

Equation 1.9 means that assigning $\mathbf{x}$ to the group having the maximum posterior probability, *i.e.*, if $q_i(\mathbf{x}) \geq q_j(\mathbf{x}),\ j = 1, 2, \ldots, g$, then $\mathbf{x}$ is assigned to group G_i. Using Bayes' rule, the posterior probability is defined as

$$\begin{aligned} q_i(\mathbf{x}_0) &= \Pr[\mathbf{x} \in G_i | \mathbf{x} = \mathbf{x}_0] \\ &= \frac{\Pr[\mathbf{x} = \mathbf{x}_0 | \mathbf{x} \in G_i] \Pr[\mathbf{x} \in G_i]}{\sum_{j=1}^{g} \Pr[\mathbf{x} = \mathbf{x}_0 | \mathbf{x} \in G_j] \Pr[\mathbf{x} \in G_j]} \\ &= \frac{f_i(\mathbf{x}_0)\pi_i}{\sum_{j=1}^{g} f_j(\mathbf{x}_0)\pi_j}. \end{aligned} \tag{1.11}$$

For $g = 2$, equation 1.11 may be rewritten as

$$q_i(\mathbf{x}_0) = \frac{f_i(\mathbf{x}_0)\pi_i}{f_1(\mathbf{x}_0)\pi_1 + f_2(\mathbf{x}_0)\pi_2}. \tag{1.12}$$

Next, the group specific densities, $f_i(\mathbf{x}), i = 1, 2, \ldots, g$, must be estimated. If the group specific densities are known, a *parametric* approach may be used. For

example, one may assume each group has multivariate normal distribution. Alternatively, a *nonparametric* method may be used. Nonparametric methods of density estimation are not based on distributional assumptions [26].

Schneidewind has discussed the usefulness of nonparametric statistical methods in modeling software engineering data [25]. In addition, our experience has confirmed a published report that software product measures are usually not normally distributed [24]. Since a parametric description is likely to be inadequate, we consider only a nonparametric model in this study. The following notation is now needed to complete our discussion on nonparametric discriminant modeling:

- $\mathbf{y}, \mathbf{z}$ are p-dimensional vectors,
- $\hat{f}_i(\mathbf{x}|\lambda)$ is an approximation of $f_i(\mathbf{x})$,
- λ is a smoothing parameter,
- $K_i(\mathbf{y}|\mathbf{z}, \lambda)$ is a kernel density function, and
- $\mathbf{S}_i$ is the covariance matrix for the sample data in G_i.

In a nonparametric model, the density, $f_i(\mathbf{x})$, is estimated from the sample data, $\mathbf{x}_{ij},\ i = 1, 2, \ldots, g$, where $1 \leq j \leq n_i$. Using the *kernel* method of multivariate density estimation [26], $f_i(\mathbf{x})$ is estimated by

$$\hat{f}_i(\mathbf{x}|\lambda) = \frac{1}{n_i} \sum_{j=1}^{n_i} K_i(\mathbf{x}|\mathbf{x}_{ij}, \lambda),$$

where $K_i(\mathbf{y}|\mathbf{z},\lambda)$ gives a kernel probability density function on $\mathbf{y}$ with mode at $\mathbf{z}$ and smoothing parameter λ. Of the many kernel functions available, we used the *normal kernel*,

$$K_i(\mathbf{y}|\mathbf{z},\lambda) = (2\pi\lambda^2)^{-n_i/2}|\mathbf{S}_i|^{-1/2}\exp\{(-1/2\lambda^2)(\mathbf{y}-\mathbf{z})'\mathbf{S}_i^{-1}(\mathbf{y}-\mathbf{z})\},$$

where $|\mathbf{S}_i|$ is the determinant of the covariance matrix $\mathbf{S}_i$. Various methodologies have been proposed for selecting the value of λ. For most applications it is sufficient to empirically determine the value of λ that best fits the observed data. Substituting into Equation 1.10 with $f_i(\mathbf{x}) = \hat{f}_i(\mathbf{x}|\lambda)$, gives the discriminant function.

1.8 Stepwise Model Selection

Prior to fitting a discriminant model, one must identify which combination of independent variables yields the best classification. This is known as model selection. Model selection selects $m \leq p$ measures, where p is the number of independent variables comprising each observation. The m measures selected each contribute to the model at a significance level determined by the analyst. In this study, stepwise discriminant analysis was used.

Stepwise discriminant analysis is an iterative procedure [26]. Independent variables are entered into the model in an incremental manner, based on an F test from analysis of variance, which is recomputed for each change in the current model. The process begins with no variables in the model. Then, we add the variable not already in the model with the best significance level, as long as its significance is

better than the threshold (5%). Then we remove the variable already in the model with the worst significance level, as long as its significance is worse than the threshold (5%). These steps are repeated until no variable can be added to the model.

1.9 Evaluation of Models

The normal rule to evaluate a model is to use the fit data set to estimate parameters of a model. The test data set is used to evaluate its accuracy. Both data sets consist of the actual class and values of the independent variables for each observation. Ideally, the test data set is an independent sample of observations.

In our study, first we use all the observations collected from the system under investigation to fit a model. The fitted model is then applied to the fitting data, classifying it into groups. The number of classification errors quantifies the models quality of fit.

We evaluate our model by evaluating the ability of the model to correctly classify new program modules into one of three groups predefined like Green, Yellow and Red, or high, medium, and low risk. A complete study will evaluate a model's classification performance within the limits of the data available for study. Quality of fit alone does not imply the model will have good predictive quality [12].

Normally, we prefer to evaluate the classification performance of a model by applying it to data collected from a similar system, a testing data set, and compare the predictions with the known results. In this case, however, data from a similar

project was not available.

Data splitting may be appropriate when data on a similar subsequent project is not available. Data splitting is often used provided the data set is large. Using this method, the data set is randomly partitioned into two sets: one to fit the model and one to test predictive quality of the model. Since our data set was not large enough to justify the data splitting technique, we chose to apply a modified data splitting technique to evaluate the classification performance of our model. This method usually is called Cross-Validation, sometimes called "U-Method" [6].

Using this method, we fitted a model, Mod_i, with all N observations except the i^{th}, $1 \leq i \leq N$, and predicted the missing observation, a testing set of size 1. Classifying the missing observation this way simulates the application of the model to a current project with unknown results. In practice, this means N different discriminant models were fitted with $N-1$ observations and each model was then used to classify the missing observation.

This technique is considered a true validation since the i^{th} observation is not used at the same time to fit the model and to assess the predictive quality. This method is appropriate for small data sets than data splitting, but involves more computation per observation. It is often used in multiple linear regression in cases where the data set is small [22].

BIBLIOGRAPHY

[1] V. R. Basili, L. C. Briand, and W. Melo. A validation of object-oriented design metrics as quality factors. *IEEE Transactions on Software Engineering*, 22(10):751–761, October 1996.

[2] B. Beizer. *Software Testing Techniques.* Van Nostand Reinhold, New York, 2nd edition, 1990.

[3] W. R. Dillon and M. Goldstein. *Multivariate Analysis: Methods and Applications.* John Wiley and Sons, New York, 1984.

[4] Norman E. Fenton and Shari Lawerence Pfleeger. *Software Metrics.* PWS Publishing Company, New York, 2d edition, 1997.

[5] M. Halstead. *Elements of Software Science.* Elsevier, New York, 1977.

[6] T. M. Khoshgoftaar and E. B. Allen. Classification of fault-prone software modules: Prior probabilities, costs, and model evaluation. Technical Report TR-CSE-97-52, Florida Atlantic University, Boca Raton, FL, June 1997.

[7] Taghi M. Khoshgoftaar, Edward. B. Allen, Robert Halstead, Gary P. Trio, and Ronald Flass. Process measures for predicting software quality. *Computer*, 31(4):66–72, April 1998.

[8] Taghi M. Khoshgoftaar, Edward. B. Allen, Kalai S. Kalaichelvan, and Nishith Goel. The impact of software evolution and reuse on software quality. *Empirical Software Engineering: An International Journal*, 1(1):31–44, 1996.

[9] Taghi M. Khoshgoftaar, David L. Lanning, and Abhijit 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, February 1994.

[10] Taghi 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, February 1990.

[11] Taghi M. Khoshgoftaar and Robert M. Szabo. Predicting software quality during testing using neural network models: A comparative study. *International Journal of Reliability, Quality, and Safety Engineering*, 1(3):303–319, September 1994.

[12] Taghi M. Khoshgoftaar, Robert M. Szabo, and Timothy G. Woodcock. An empirical study of program quality during testing and maintenance. *Software Quality Journal*, 3(3):137–151, September 1994.

[13] T.M. Khoshgoftaar, E.B. Allen, and J.C. Busboom. Modeling software quality: The software measurement analysis and reliability toolkit. *In Processings of the Twelfth IEEE International Conference on Tools with Artificial Intelligence*, pages 54–61, Nov. 2000.

[14] B. A. Kitchenham, L. M. Pickard, and S. J. Linkman. An evaluation of some design metrics. *Software Engineering Journal*, 5(1):50–58, 1990.

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

[16] M. R. Lyu. Introduction. In M. R. Lyu, editor, *Handbook of Software Reliability Engineering*, chapter 1, pages 3–25. McGraw-Hill, New York, 1996.

[17] T. J. McCabe. A complexity metric. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, December 1976.

[18] J. C. Munson and T. M. Khoshgoftaar. The dimensionality of program complexity. In *Proceedings of the Eleventh International Conference on Software Engineering*, pages 245–253, Pittsburgh, PA, May 1989. IEEE Computer Society.

[19] J. C. Munson and T. M. Khoshgoftaar. Some primitive control flow metrics. In *Proceedings of the Annual Oregon Workshop on Software Metrics*, Silver

Falls, OR, March 1991. Oregon Center for Advanced Technology Education, Portland State University.

[20] J. C. Munson and T. M. Khoshgoftaar. The detection of fault-prone programs. *IEEE Transactions on Software Engineering*, pages 423–433, May 1992.

[21] J. C. Munson and T. M. Khoshgoftaar. The detection of fault-prone programs. *IEEE Transactions on Software Engineering*, 18(5):423–433, May 1992.

[22] R. H. Myers. *Classical and Modern Regression with Applications.* Duxbury Press, Boston MA, 1990.

[23] R. S. Pressman. *Software Engineering: A Practitioner's Approach.* McGraw-Hill, New York, 3rd edition, 1992.

[24] P. N. Robillard and D. Coupal. Study on the normality of metrics distributions. *Proceedings of the Annual Oregon Workshop on Software Metrics*, March 1991.

[25] N. F. Schneidewind. Methodology for validating software metrics. *IEEE Transactions on Software Engineering*, 18(5):410–422, May 1992.

[26] G. A. F. Seber. Multivariate observations. *IEEE Computer*, 1994.

[27] M. Shepperd. Early life-cycle metrics and software quality models. *Journal of Information and Software Technology*, 32(4):311–316, 1990.

[28] Robert M. Szabo and Taghi M. Khoshgoftaar. Classifying software modules into three risk groups. In Hoang Pham and Ming-Wei Lu, editors, *Proceedings: Sixth ISSAT International Conference on Reliability and Quality in Design*, Orlando, Florida USA, August 1999. International Society of Science and Applied Technologies. Invited paper. In press.

[29] N. Wirth. A plea for lean software. *IEEE Computer*, 28(2):64–68, February 1995.