The Detection of Fault-Prone Program Using a Neural Network

Shuji Takabayashi¹, Akito Monden¹, Shin-ichi Sato², Ken-ichi Matsumoto¹, Katsuro Inoue¹,³ and Koji Torii¹

¹Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0101, Japan
²NTT DATA Corporation, Laboratory for Information Technology, Department of Research & Development Headquarters
Kayabacho-Tower, 21-2, Shinkawa 1-chome, Chuo-ku, Tokyo 104-0033, Japan
³Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan
{shuji-t, akito-m}@is.aist-nara.ac.jp, s-sato@rd.nttdata.co.jp, {matumoto, k-inoue, torii}@is.aist-nara.ac.jp

ABSTRACT

This paper proposes a discriminant analysis method that uses a neural network model to predict the fault-prone program modules that will cause failure after the release. In our method, neural networks of a layered type are used to represent nonlinear relation among predictor variables and objective variables. Since the relation among predictor variables and objective variables is complicated in real software, linear representation used in conventional discriminant analysis is not suitable for the prediction model. To evaluate the method, we have measured 20 metrics, as predictor variables, from a large scale software that have been maintained more than 20 years, and also measured the number of faults found after the release as objective variables. Result of the evaluation showed that prediction accuracy of our model is better than that of conventional linear model.

KEYWORDS
discriminant analysis, AIC, neural network

INTRODUCTION

Many companies have legacy software that had been developed many years ago and have been continuously modified and expanded till today. However, continuous modification and expansion in large scale software produce many complex and fault-prone program modules, that will increase the maintenance cost [6]. In order to lessen such maintenance cost, we need to predict the fault-prone modules in advance, and to test them thoroughly or sometimes even restructure them as new modules.

To detect the fault-prone modules, we need to construct a discriminant model, that has multiple predictor variables such as complexity measures of software, to predict which module is faulty. Conventional linear discriminant model is as follows [3]:

\[ P = \sum_{j \in D} a_j N_j + C \]

\[ \begin{cases} \text{if } P > 0 & \text{: classified as "modules which don't include any faults" } \\ \text{if } P < 0 & \text{: classified as "module which include at least one fault" } \end{cases} \]

Where D denotes a set of subscript of predictor variables that are chosen out from variables that we measure, \( N_i \) ( \( i = 1, \ldots, n \) ) denotes an observation value, \( a_i \) ( \( i = 1, \ldots, n \) ) denotes a distinction coefficient, and C denotes a constant.

However, this linear model is not suitable for the prediction model. In this model, there is an assumption that the effect of each factor to the software reliability is additive, i.e., independent of the other factors. It is, however, not plausible because the factors are mutually and complicatedly related to each other in real software. We
need a nonlinear analysis method using a nonlinear representation model.

In this paper, we propose a nonlinear discriminant analysis method that uses a neural network model to predict the fault-prone modules in large scale software. In the method, neural networks of a layered type are used to represent nonlinear relation among predictor variables and objective variables. A layered neural network is a data processing model that represents nonlinear relation among the inputs and the outputs by simple processing units and weighted links connecting the units.

To construct a prediction model, we need to know the most appropriate combination of predictor variables. This paper presents a method for selecting a combination of predictor variables that strongly affect the existence of faults in programs. To select predictor variables, we use a forward selection method [3], in which variables are added to a prediction model one by one. To evaluate prediction models that have different numbers of predictor variables, a statistical information criterion AIC (Akaike’s Information Criterion) was used [1].

Takada et al.[7] also proposed a software reliability prediction model that uses a neural network. However, their model evaluates the reliability of a software development project itself and predicts the number of faults during the development, while our model evaluates the reliability of each program module and discriminates the fault-prone modules after the release.

In what follows, we first propose a nonlinear model construction method, comparing it with a linear model. Then describes the data used for the experiment, and explains how models were constructed in the experiment. Next, we discuss on the accuracy of linear and nonlinear models and lessons learned on the experiment. Finally the paper describes the conclusion.

MODEL CONSTRUCTION METHOD

The proposed nonlinear model construction method is an extension of a linear model construction method. In this section, first we describe the linear method and then describe the nonlinear method.

Method for linear discriminant models

In order to construct a liner discriminant model, we take advantage of the relation between discriminant statistical analysis and multivariate regression analysis. In multiple regression analysis, which is a multivariate analysis method [3], the relation among predictor variables and objective variables is represented by a linear equation, such as:

\[ Y = a_0 + a_1 N_1 + \ldots + a_i N_i + E \]

Where \( Y \) denotes the objective variable, \( N_i \) (\( i = 1, \ldots, n \)) denotes a predictor variable, \( a_i \) (\( i = 0, \ldots, n \)) denotes a coefficient, and \( E \) denotes the residual between the predicted value and the actual value.

In order to discriminate the program modules into two groups, regression analysis can be used, as shown in figure 1. In figure 1, the first group contains program modules that have a fault, and the second group contains modules that have no fault. \( N_1, N_2, Y_1 \) and \( Y_2 \) are defined as follows:

\[ N_1 = \text{the number of modules belong to the first group.} \]
\[ N_2 = \text{the number of modules belong to the 2nd group.} \]
\[ Y_1 = N_2 / (N_1 + N_2) \]
\[ Y_2 = N_1 / (N_1 + N_2) \]

![Figure 1. Relation between discriminant analysis and regression analysis](image-url)
In the model construction phase, in case a module belongs to the first group, the value of the objective variable \( Y \) is set to \( Y_1 \); and, in case a module belongs to the second group, \( Y \) is set to \( -Y_2 \). Since \( N_1 \) and \( N_2 \) are not equal, we assign a weighted value.

In the discriminant phase, the discriminant value \( Y \) obtained by this regression analysis is interpreted as follows:

- \( Y > 0 \): classified as first group.
- \( Y < 0 \): classified as second group.

Let us assume that objective variables and related variables are given. The method of selecting an appropriate combination of predictor variables and constructing a prediction model as follows:

1. **Step1.** Construct 1-input prediction models. Models as many as the variables are constructed. Use a least square method to estimate model parameters.
2. **Step2.** Select the best models among the constructed models based on AIC, a statistical criterion.
3. **Step3.** Construct new models by adding one more predictor variable to the temporary best model. Models as many as the remaining variables are constructed. Then return to Step2.

Step 2 and 3 are repeated until no variable remains or the goodness of the temporary best model becomes worse than that of the less variable model. Finally the best model among all of the constructed models is selected.

As the statistical criterion of goodness of models, we use AIC. AIC means the minus double of the logarithmic likelihood of a model against given data:

\[
AIC = (\text{Number of samples}) \log (\text{Residual sum of squares}) + 2(\text{Number of parameters})
\]

In the above, the first term is the minus double of an estimate of the logarithmic likelihood, and the second term is the expected difference between the real logarithmic likelihood and the estimate. The estimate of the first term is biased. It tends to become larger that the real logarithmic likelihood when the number of parameters is large. The second term has the role of compensating the bias. Therefore, we can use AIC to evaluate models that have difference numbers of parameters.

**Method for nonlinear discriminant models**

We use a neural network model of three layers (Input layer, Intermediate Layer, and Output layer) as a nonlinear discriminant model. We fixed the number of intermediate layer units to 3. Predictor variables \( N_i \) are input to units on input layer; and, the objective variable \( Y \) is output from the unit on output layer. Each unit of input layer receives the value of a predictor variable and sends the weighted value to the intermediate layer. In the intermediate layer and the output layer, each unit receives the weighted values from the beyond layer and sum them up. The unit then nonlinearly transform the sum by a sigmoid function \( g(\beta) = \frac{2}{1+\exp(-\beta)}-1 \). A neural network, which is composed of multiple processing units, is able to represent complicated relation among predictor variables and an objective variable.

The major difference from the linear model construction model is that model parameters are estimated by a learning algorithm.

1. **Step1.** Construct neural network. Then, let them learn the relation between the objective and predictor variables. Models as many as the variables are constructed.
2. **Step2.** Select the best models among the constructed models based on AIC.
3. **Step3.** Construct new models by adding one more predictor variable to the temporary best model selected in Step2, and let it learn the relation again. Models as many as the remaining variables are constructed. Then return to Step2.

To determine the link weights appropriately from samples of predictor and objective variables, we use a standard learning algorithm called error backpropagation algorithm [5].
DATA FOR THE EXPERIMENT

The Data we used for the experiment are collected from large scale program developed by Japanese software company. This program has been maintained for about 20 years and modified and expanded many times during that period. The program was written in an old programming language (called HPL) that is peculiar to the hardware. The number of faults and 20 kind of metrics are measured in each file (module). In this case, the number of faults means the number of modifications done for correcting defects. The numbers of program modules we measured is 1436.

We classified the collected metrics into three groups from the difficulty of measuring. Since some metrics are not easy to measure, software developers cannot always collect all the metrics. Here, we classify metrics into 3 groups as follows:

- **Group T**: Set of metrics that can be measured by lexical analysis (easy to measure).
- **Group P**: Set of metrics that can be measured by syntactical analysis (more difficult to measure).
- **Group O**: Set of metrics that cannot be measured from source code.

Actual metrics in each group are described below:

**Group T (Metrics obtained by Token analyzer)**

T1: Lines of code.
T2: The number of comment lines.
T3: The number of procedure-call statements and jump statements.
T4: The number of the modules called from the target module.
T5-T9: Halstead’s Software Science [6].
   - T5: Vocabulary.
   - T6: Length.
   - T7: Volume.
   - T8: Difficulty.
   - T9: Effort.

**Group P (Metrics obtained by Parser)**

P1: Cyclomatic number [6].
P2: The sum of nest level of each statement.
P3: The numbers of loop nodes.
P4: The number of substitution variables.
P5: The number of reference variables.
P6: The number of external substitution variables.
P7: The number of external reference variables.
P8: The sum of level of each substitution variable.
P9: The sum of level of each reference variables.

**Group O (Other metrics)**

E1: Version number.
E2: Lapsed days from the first release.

EXPERIMENT

We have conducted an experiment to evaluate the performance of our nonlinear model, comparing it with a linear model.

At first, we divided 1436 program modules into two groups at random. Each group has 718 modules. The first group is used for estimating the model parameters; and, the second group is used for evaluating the constructed models.

Then, we divided each group into two more groups: fault-free group and fault-prone group. Fault-free group contains modules in which no fault was found after the release. Fault-prone group contains modules in which more than one fault was found after the release. The rate of the program in which the bug was actually contained is about ten percent of the whole.

In the experiment, we constructed two types of models, one is a model using the linear discriminant analysis, and the other is a model using neural network. Furthermore, following three types of model were constructed in each model. Therefore, we have six types of model in total.

- Model A: Uses predictor variables of group T.
- Model B: Uses predictor variables of group T and P.
- Model C: Uses predictor variables of group T, P, and O.

Table 1 and 2 show the variables selected by a forward selection method, and the value of AIC measured for each model. Variables are added to the model as long as the value...
of AIC decreases. For example, in Linear Model A, a metric T5 (Halstead’s Software Science Vocabulary) is selected at first. The value of AIC is 2523.99. Next, a metric T8 (Halstead’s Software Science Difficulty) is selected. The value of AIC is 2510.19. This means two-variable model (using T5 and T8) is statistically better than one-variable model (using T5). As we increase the number of variables, value of AIC decreases. And in six-variable model, the value of AIC becomes the minimum. That is, six variables (metrics) are selected as a predictor variable of a model from eight metrics of group T.

We use three types of criterion (type I error, type II error, and accuracy) to evaluate the goodness of prediction models. Type I error is a percentage of cases where the model concludes that a module is fault-prone though it is not so in fact. Type II error is a percentage of cases where the model says that a module is fault-free though it is not so. Accuracy is a percentage of cases where the model distinguishes modules correctly.

The results are showed in Table 3. Table 3(a) shows the residual variance (prediction error) when we apply the model to evaluation data. Table 3(b) shows type I error, type II error and accuracy.

**DISCUSSION**

As shown in the Table 3(a), we see that prediction error of nonlinear model has been improved as compared with the linear model. In the linear model, residual of Model C is the smallest. In the nonlinear model, the residual
of Model B and Model C is almost the same and they are better than that of Model A.

Let us compare the accuracy between linear model and nonlinear model (see Table 3(b)). Obviously, nonlinear models show good results. In the nonlinear model, the accuracy of model A shows the best. Therefore, if we use the nonlinear model, we can obtain enough accuracy by collecting only few metrics, which can be measured by lexical analysis. So the effort of software developers will be lessened by use of the nonlinear model.

As shown in table 3, type I error is improved at nonlinear model. On the other hand, type II error is not improved. However, this does not result in that linear model is a good model. There are tradeoffs between type I error and type II error. Linear models tend to discriminate many programs as fault-prone programs, and this causes the improvement of type II error while it causes the corruption of type I error.

In this experiment, type I errors are not so good on the whole. This is because the data we used for the experiment had a unique character. We have counted the number of fault found in the programs after the release. Therefore, not so many faults were found. The rate of programs in which fault was found is less than 10 %. Hence, when we found only one fault, we considered as fault-prone program. So we couldn't enhance the difference of characteristics between the sound programs and dangerous programs.

CONCLUSION

We have presented a method for systematically constructing nonlinear models that discriminate fault-prone programs. Through an experiment of real software developments, we have shown that the proposed model shows better performance in discrimination than conventional discriminant analysis.

There are some other related researches reported. Munson et al.[4] also proposed a nonlinear model in which they employ a principal-components procedure to reduce predictor variables. In the future, we are going to compare the goodness between their model and our model.

REFERENCES