Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 49 (2007) 483–492
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Object-oriented software fault prediction using neural networks

S. Kanmani [a,*], V. Rhymend Uthariaraj [b], V. Sankaranarayanan [c], P. Thambidurai [a]

[a] Department of Computer Science and Engineering, Pondicherry Engineering College, Pondicherry 605014, India
[b] Department of Information Technology, Anna University, MIT Campus, Chennai, India
[c] Tamil Virtual University, Chennai, India

Received 30 November 2005; received in revised form 14 July 2006; accepted 24 July 2006
Available online 6 September 2006

Abstract

This paper introduces two neural network based software fault prediction models using Object-Oriented metrics. They are empirically validated using a data set collected from the software modules developed by the graduate students of our academic institution. The results are compared with two statistical models using five quality attributes and found that neural networks do better. Among the two neural networks, Probabilistic Neural Networks outperform in predicting the fault proneness of the Object-Oriented modules developed.

Keywords: Object-Oriented metrics; Fault proneness; Back propagation neural network; Probabilistic neural network; Discriminant analysis; Logistic regression

1. Introduction

The last decade has witnessed the rise to prominence of the Object-Oriented (OO) paradigm. In this paradigm, new proposals in the areas of the programming languages, methodologies, tools, metrics, etc. have grown up. Many researchers proposed specific metrics for OO paradigm [1,3–5,8,13,18,19], and still are being proposed. The usefulness of these OO metrics for the prediction of the software module quality is validated by empirical studies [2,3,6,10,12,16,20]. These studies used quality attributes as the dependent variables and OO metrics as the independent variables.

In practice, quality estimation means estimating reliability, maintainability, understandability, etc. Reliability is typically measured by the defects/faults in the software modules. Predicting (or classifying) the fault prone software module is a challenge for the developers before the software is released. Hence accurate prediction models are experimented in the literature, so as to revise the predicted fault prone modules to attain 100% defect-free software.

Several researchers worked on the development of prediction models for the classification of fault prone modules. Briand et al. [6] carried out an empirical validation of OO measures to understand their relationships with fault proneness using univariate and multivariate logistic regression models. Tang et al. [23] investigated the correlation between Chidamber and Kemberer (CK) metrics and three types of OO faults.

Emam et al. [11] reported that export coupling has a strong association with fault proneness using a commercial Java Application. Fioravanti and Nesi [12] identified 19 influential measures among the 200 considered by a stepwise process and developed three models (first with size measures, second with OO measures and third with all of the measures). Denaro and Pezze [10] concluded that software faultiness across homogeneous applications could be predicted from an empirical study with two versions of Apache Web server. Most of these prediction models are developed using statistical methods.

* Corresponding author. Tel.: +91 413 2655281;Fax: +91 413 2655101.
E-mail address: kanmani.n@gmail.com (S. Kanmani).

0950-5849/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2006.07.005

Neural networks have seen an explosion of interest over the years, and are being successfully applied for the problems of prediction, classification or control. Khoshgoftaar et al. [14,17] experimented neural network based classification model in telecommunication software products and found that the model is more accurate than discriminant model. In their work, they used nine metrics for the modules developed in the procedural paradigm.

In this work, two neural networks, Back Propagation Neural Network (BPN) and Probabilistic Neural Network (PNN) are introduced for predicting the fault proneness of the C++ modules (classes) using OO class metrics. The results obtained by these models are compared with that of the statistical methods (discriminant analysis and logistic regression) by computing five model quality parameters. To analyze the relationship clearly three models are developed in each method: using only size measures, using OO measures (meant for coupling, cohesion and inheritance) and using all of them. To carry out this investigation, the OO modules (in C++) developed by the students in our institution for the general library maintenance process is used.

This paper is outlined as follows. Section 2 discusses the methodology used to develop the models. The various methods used to model the prediction system are introduced in Section 3. The case study is reported in Section 4. Section 5 summarizes the results and Section 6 narrates the related work.

2. Model methodology

We used the following methodology to develop the prediction models with three categories of measures – Size only (Model I), Object-Oriented metrics only (Model II) and both (Model III).

Suppose we have measurements on a sample of n modules, using a set of m metrics, the following is the summary of our methodology for developing and validating the prediction models.

i. Standardize the measurements to a mean of zero and a variance of one for each metric. ($Z(n \times m)$, n is the number of modules, m is the number of metrics).
ii. Perform principal component analysis on the standardized metrics to produce domain metrics.
iii. Prepare the training (fit) and testing (validation) data sets.
iv. Develop the models (I, II and III) using the respective training data set.
   (a) Develop discriminant model,
   (b) Develop logistic regression model,
   (c) Develop the neural network model using BPN,
   (d) Develop the neural network model using PNN.
v. Predict the class (fault or non-fault) of each module in the test data set.
vi. Compare the prediction results with actual values. Compute the quality parameters – misclassification rates, correctness, completeness, effectiveness and efficiency, and compare the prediction accuracy of the four methods.

2.1. Principal component analysis

Software metrics are often highly correlated with one another, because the attributes of software components are often related. Unfortunately, if the independent variables of a model are highly correlated, estimates of the model parameters may not be stable. In other words, insignificant variations in the data may result in drastically different parameter estimates. Principal Components Analysis (PCA) is the widely used statistical method [22] to reduce the effect of these correlations. Principal components analysis can transform the original set of correlated variables into a smaller set of uncorrelated variables that are linear combinations of the original ones for most robust modeling. The new principal components variables are referred as *domain metrics*.

Assuming $Z(n \times m)$ matrix as the standardized metric data with n software modules and m measurements, the principal components algorithm works as follows.

- Calculate the covariance matrix $\mathbf{\Sigma}$ of $\boldsymbol{Z}$.
- Calculate eigen values λ_j and eigenvectors $\mathbf{e}_j$ of Σ, $j = 1,\ldots,m$.
- Reduce the dimensionality of the data (using a cutoff for the percentage of variance in the data to be considered or for the eigen values).
- Calculate a standardized transformation matrix $\boldsymbol{T}$ where each column is defined as

$$\mathbf{t}_j = \frac{\mathbf{e}_j}{\sqrt{\lambda_j}} \text{ for } j = 1, \ldots, p \tag{1}$$

- Calculate domain metrics for each module using

$$D_j = \mathbf{Z}t_j \tag{2}$$

$$\boldsymbol{D} = \boldsymbol{Z}\mathbf{T} \tag{3}$$

The final result $\boldsymbol{D}$ is an $n \times p$ matrix of domain metrics with a mean of zero and variance of one, and each row is a set of domain metric values for a module.

2.2. Evaluating the accuracy of the model

The validity of the prediction models varies greatly. Many approaches for evaluating the quality of the prediction models are used. One among them is data splitting. The data set is divided randomly into training set and test set. The model is constructed using training set and used in the prediction of fault proneness for the test data set. In this work the results obtained for the test data set are evaluated using the following model quality parameters.

2.2.1. Misclassification rates

Misclassification rate is defined by Khoshgoftaar et al. [17] as the ratio of *number of wrongly classified modules* to the *total number of modules* classified by the prediction system. The wrong classifications fall into two categories. If non-fault prone modules are classified as fault prone (C1), it is named as *Type I* error. If fault prone modules are classified as non-fault prone (C2), it is named as *Type II* error.

$$\text{Type I error} = \frac{C1}{\text{Total no. of faulty modules}} \tag{4}$$

$$\text{Type II error} = \frac{C2}{\text{Total no. of non-faulty modules}} \tag{5}$$

$$\text{Overall misclassification} = \frac{C1 + C2}{\text{Total no. of modules}} \tag{6}$$

From the results of the prediction system if a module is fault prone, the developers revise those modules and redesign. So *Type I* errors incur only modest cost in terms of extra attention to those modules. *Type II* errors risk unexpected problems late in the development as the faults may persist even after release.

2.2.2. Correctness

Correctness is defined by Briand et al. [6] as the ratio of the *number of modules correctly classified as fault prone* to the *total number of modules classified as fault prone*. Low correctness means that a high percentage of the classes being classified as fault prone, which do not actually contain any fault. So there is a waste of developers' effort in correcting them. Hence, correctness is expected to be high always.

2.2.3. Completeness

Assume that the prediction model selects classes that are classified as fault-prone for inspection and inspections are 100% effective, i.e., all faults in a class are found during inspection. Completeness is defined by Briand et al. [6] as the ratio of number of faults in classes classified as fault prone to the total number of faults in the system. It is a *measure of the percentage of faults that would have been found* if we used the prediction model in the stated manner. Counting the percentage of faults is more precise than counting the percentage of fault-prone classes. It ensures that detecting a class containing many faults, contributes more to completeness than detecting a class with only one fault.

2.2.4. Effectiveness and efficiency

Effectiveness is defined by Yuan et al. [25] as the proportion of fault prone modules considered high risk out of all modules that received it.

Let,
Type I misclassification is $Pr(fp/nfp)$
Type II misclassification is $Pr(nfp/fp)$
$Pr(fp/fp)$ is number of modules predicted and in actual are fault
π_{nfp} be the expected proportion of non-fault prone modules
π_{fp} be the expected proportion of fault-prone modules
Then

$$\text{Effectiveness} = Pr(fp/fp) = 1 - Pr(nfp/fp) \tag{7}$$

Efficiency is defined by Yuan et al. [25] as the proportion of predicted fault-prone modules that are inspected out of all modules.

$$\text{Efficiency} = \frac{\Pr(fp/fp)\pi_{\text{fp}}}{\Pr(fp/nfp)\pi_{\text{nfp}} + \Pr(fp/fp)\pi_{\text{fp}}}. \tag{8}$$

3. Methods

Many methods for developing multivariate models such as discriminant analysis, decision trees, rough set analysis, neural networks, linear and logistic regression are introduced. This section discusses the methods used in this work to develop the prediction system. The two statistical approaches (Discriminant Analysis and Logistic Regression) are employed using SPSS V7.2 package. The neural network approaches (BPN and PNN) are employed using MATLAB V6.1 with neural network toolbox V 2.0.

3.1. Discriminant analysis

Discriminant analysis estimates a rule to assign an observation to a class, minimizing the probability of misclassification [22]. We apply non-parametric discriminant analysis to classify modules as non-fault prone or fault prone. The domain metrics, D_j, are independent variables and the group membership (fault, non-fault) is the dependent variable. We estimate a discriminant function based on the training data set, and apply it to the test data set for evaluation. Consider the following notation:

- D_i is the vector of the ith modules' domain measurements
- G_1 and G_2 are mutually exclusive group fault prone and non-fault prone, respectively
- n_k is the number of modules in group G_k, $k = 1$ or 2.
- p_k is the proportion of training modules in G_k.
- $f_k(d_i)$ is the multivariate probability density giving the probability that a module d_i is in G_k. Let $\hat{f}_k(d_i/\lambda)$ be an approximation of $f_k(d_i)$ using multivariate normal kernel density estimation technique where λ is a smoothing parameter.

To minimize the probability of misclassification, the estimated discriminant function is given by,

$$\text{Assign } d_i \text{ to} \begin{cases} G_1, & if \frac{\hat{f}_1(d_i)}{\hat{f}_2(d_i)} > \frac{p_2}{p_1} \\ G_2, & \text{Otherwise} \end{cases} \tag{9}$$

3.2. Logistic regression

Logistic regression is a standard technique based on maximum likelihood estimation. A multivariate logistic regression model is developed based on the following equation.

$$p(X_1, X_2, \ldots, X_n) = \frac{e^{(C_0 + C_1X_1 \ldots + C_nX_n)}}{1 + e^{(C_0 + C_1X_1 \ldots + C_nX_n)}} \quad (10)$$

Where, the function p is the probability that a fault is found in a class and X_i's are the design measures included as the independent variables in the model (also called as the co-variates of the logistic regression equation). Logistic regression is used when the dependent is dichotomy and the independents are of any type. The coefficient C_i's are estimated through the maximization of a likelihood function.

The first step in discriminant and logistic methods is identifying which combination of independent variables best estimates the dependent variable. This is known as model selection. The techniques available for model selection include forward selection, stepwise method and backward elimination. We used stepwise method in SPSS for both discriminant and logistic regression approaches and developed the three models in each. We have chosen the classification cutoff as 0.5 for the model selection. The threshold values for the domain measures to enter and exit the model are fixed at 0.5 and 0.10, respectively. The test data was supplied to the models and the classification results are noted.

3.3. Back propagation neural network (BPN)

An artificial neural network is composed of computational processing elements with weighted connections. We used feed forward multilayer perceptron network and the back propagation training algorithm (hence referred as Back Propagation Neural Network – BPN). The neural network architecture is designed using Matlab neural network tool box V 2.0 [15].

The input layer has one neuron for each of the input variable (domain measures – D_j, $j = 1,\ldots,p$). We used one hidden layer. There are two neurons in the output layer, one for each class (fault or non-fault). The output unit with the greatest value indicates the class selected by the network. The network learns by finding a vector of connection weights and minimizes the sum of squared errors on the training data set. One pass through all of the training observations (Training Phase) is called an *epoch*. The network is trained with a continuous back propagation learning algorithm; the weights are adjusted after each observation is fed forward. Various neural network architectures are tested. The number of units in the hidden layer, learning rate (η) and momentum rate (α) are adjusted to find a preferred combination. Table 1 summarizes the architectural details of the three models developed.

The activation function used is '*tansig*'. Activation function is the logistic function, with a gain parameter that controls how sharply the function changes from zero to one. We trained the network, where the weights are adjusted after each epoch. Various values for the number of neurons are tested to find the final value. After training, the network is simulated for the validation data set (Testing Phase) and the classification outputs are obtained.

3.4. Probabilistic neural network (PNN)

The probabilistic neural network was developed by Specht [21]. This network provides a general solution to pattern classification problems by following an approach developed in statistics, called Bayesian classifiers. The probabilistic neural network uses a supervised training set to develop distribution functions within a pattern layer. These functions are used to estimate the likelihood of an input feature vector being part of a learned category, or class.

PNN is based on one pass learning with highly parallel structure. It is a powerful memory based network and able to deal with sparse data effectively. In PNN, the number of neurons in the hidden layers is usually the number of patterns in the training set because each pattern in the training set is represented by one neuron. The main advantage of PNN is the speed at which the network can be trained. Training a PNN is performed in one pass. The smoothing factor allows PNN to interpolate between the patterns in the training set.

Using the Matlab neural network tool box, the PNN was developed for the three models with 790 (size of the training data set) neurons in the hidden layer and with the respective domain metrics as the input. The smoothing factor is fixed at 1 for all of the three models. After training the network with a single pass, the test set is inputted and the classification results are obtained.

4. Case study

The software system used for this study is developed by the Graduate students of the Department of Computer Science and Engineering at Pondicherry Engineering College.

Table 1
BPN parameters

Model No.	No. of neurons in three layers	Learning rate η	Momentum rate α	Gain	Epoches
I	1, 5, 2	0.1	0.4	1	346
II	6, 19, 2	0.1	0.4	1	371
III	6, 19, 2	0.1	0.4	1	368

All students had experience with C and C++ programming languages. The students are grouped into 200 teams with a maximum of three in each team. Each team is asked to develop a Departmental Library Management System that supports the issue and return process of books to staff members and students. They developed the system in Turbo C++ environment with necessary libraries, using files for data storage.

The total number of classes (modules) in the resulted 200 software systems is 1185. Each of the classes contains a number of attributes, a number of methods and a number of statements. In terms of these, the descriptive statistics of the classes used in the experiment are given in Table 2. It shows the maximum value, interquartile ranges, median, minimum and mean values. From the data it is found that less number of attributes is defined in the modules, even though the maximum value is 29.

A tool named "Object Oriented Metric Calculator" (OOMC) is developed and used to compute the values of the various OO measures from the C++ source code automatically.

4.1. Independent variables

In order to carry out this experiment, the OO measures (cohesion 10, inheritance 18, coupling 29, size 7 and totally 64) already used (Briand et al. [6,7]) in the literature are considered. Using OOMC all metric values are computed. As recommended by Briand et al. [6], the measures with maximum number of zeroes are eliminated from the study. They are NMNpub, SPD, IFCMIC, ACMIC, FCMEC, DCMEC, IFMMIC and FMMEC (8 nos.).

As the highly correlated measures affect the performance of the prediction model, the correlation between the 56 metrics are found out by Pearson correlation co-efficient and analyzed. The measures with high correlation co-efficient (>8.5) are removed. Finally the number of measures are reduced to 18 (3 inheritance measures, 6 coupling measures, 6 cohesion measures and 3 size measures) which are given in the Table 3. Analysis of data distributions and outliers are performed as suggested in Briand et al. [6] and [22].

The size measures can only be computed from source code unlike other OO measures (inheritance, cohesion and coupling – which can be computed from design document). In order to study the influence of the size measures in the prediction, three models are developed in each of the methods – Model I with only size measures, Model II with OO measures and Model III with all of the measures.

Table 3
Reduced set of Metrics

Metrics	Description
Inheritance measures	
DIT	Depth of Inheritance Tree
CLD	Class to leaf Depth
NOC	Number of Children
Coupling Measures	
CBO	Coupling Between Objects
RFC	Response set for a class
AMMIC	Ancestor class Method–Method Import Coupling
DMMEC	Descendant class Method–Method Export Coupling
OMMIC	Other class Method–Method Import Coupling
OMMEC	Other class Method–Method Export Coupling
Cohesion measures	
LCOM1	Lack of cohesion among methods
LCOM4	Variation in LCOM1
CO	Connectivity
LCOM5	Variation in LCOM4
Coh	Cohesion
TCC	Tight Class Coupling
Size measures	
NM	Number of Methods implemented
NA	Number of Attributes implemented
Stmts	Number of statements

Using SPSS, the principal components for each of the models (I, II, and III) with their respective metrics (size 3, OO 15 and all 18) are identified. The stopping rule used is 'eigen value $\geqslant 1$'. Due to this stopping rule the number of principal components for the models I, II and III are cut down to 1, 6 and 6, respectively. The principal components obtained for each of the models are given in Tables 4–6. One component with 50.7% variance is obtained for Model I. Six components with a cumulative variance of 73.5% and 66.9% are obtained for Model II and Model III, respectively. Due to the stopping rule chosen, the percentage of variance is less.

4.2. Dependent variable

Fault proneness is considered as the quality attribute (dependent variable) to be predicted by the prediction model. A team of three experienced staff members of the department carried out testing of the developed software systems. This group derived about 50 major test cases for the given problem. They mapped the test cases to the individual classes by analyzing their responsibilities. On execution of the software, appropriate input values are given for each of the test cases and tested. If any of the test cases is

Table 2
Size details of the modules developed in the experiment

Metrics	Max.	75%	Median	25%	Min.	Mean
No. of Statements	1576	141	75	24	1	111.3
No. of Methods	27	7	4	2	0	5.13
No. of Attributes	29	2	0	0	0	1.77

Table 4
Principal component for Model I

Metrics	PC1
Stmts	0.672
NA	0.675
NM	0.783
Eigen value	1.520
% Var.	50.662
% Cum. Var.	**50.662**

not satisfied, the module corresponding to that is declared 'fault'. If all of the test cases mapped for the class are satisfied, it is declared 'non-fault'. In order to compute the completeness, the number of test cases failed for each class is calculated, which gives the number of faults in that class. Among the 1185 classes in the experiment, 475 classes are identified as faulty classes (40%) and 710 classes are identified as non-faulty classes (60%).

5. Results and discussion

From the 1185 data set we randomly selected two third of the classes (790) for the training set. The test set consisted of the remaining 395 classes. Among the 790 classes in the training set, 317 classes are found with fault and 473 are without any fault. In the test set, 158 classes are with fault and the rest 237 are non-fault. The prediction systems are developed using the training set in each of the methods discussed in Section 3 for the three models I, II and III. The test set domain metrics are supplied to the prediction systems and output (the fault proneness results) are collected. The results are compared to the actual fault proneness values and the five quality parameters are computed.

Tables 7–10 summarize the misclassification results. G_1 refers to the fault group and G_2 refers to the non-fault group. The four possible output results are presented in the inner matrix of the tables, which are G_1G_1 (actual fault and predicted fault), G_1G_2 (actual fault and predicted non-fault – *Type II error*), G_2G_1 (actual non-fault and predicted fault – *Type I error*) and G_2G_2 (actual non-fault and predicted non-fault). The outer matrix shows the classification results of fault proneness and non-fault proneness for actual and the prediction systems. The overall misclassification is given at the bottom of the matrix.

Table 7 presents the results obtained by the discriminant analysis. Due to the reduction in *Type I error*, Model III has better performance. Table 8 shows the results of logistic regression method. Even though the *Type II errors* are comparatively lesser than the discriminant models, (which is advantageous) the overall misclassification is high due to the high values of *Type I error*.

Table 9 presents the results of BPN prediction. The error rates of the models are very much lesser than the two statistical models. Table 10 gives the results of PNN prediction. The overall misclassification rate is reduced considerably. This represents the high accuracy in prediction by these models.

The summary of misclassification in all prediction systems is depicted in Fig. 1. It shows the better and consistent performance of PNN based models. Among the three models, the compound effects of size and OO metrics (Model III) give high prediction results.

The correctness is computed as the ratio of the actual fault prone classes (the entry G_1G_1 of the Tables 7–10) to the total fault-prone classes (the column total of predicted G_1 in the Tables 7–10), identified by the model. From the results in Table 11, it is found that the logistic regression- and discriminant- based models lead to low correctness, which implies that a large number of classes that do not contain fault would have been inspected. Even though

Table 5
Principal components for Model II

Metrics	PC1	PC2	PC3	PC4	PC5	PC6
DIT	0.049	−0.589	0.235	0.207	0.443	0.294
CLD	−0.329	0.425	−0.203	0.515	0.047	−0.091
NOC	−0.306	0.630	−0.203	0.448	0.243	−0.066
CBO	0.450	−0.252	0.281	0.194	0.097	−0.353
RFC	0.803	0.182	0.407	0.310	−0.088	0.086
AMMIC	0.197	−0.297	0.435	−0.048	0.664	−0.040
OMMIC	0.680	0.164	0.342	0.423	−0.384	0.083
DMMEC	−0.161	0.455	0.000	0.373	0.451	0.231
OMMEC	0.108	0.258	−0.031	−0.476	0.012	0.623
LCOM1	0.732	0.370	−0.076	−0.082	0.061	0.088
LCOM4	0.453	0.362	−0.256	−0.432	0.347	−0.298
CO	−0.183	0.452	0.458	−0.130	−0.034	0.324
LCOM5	0.439	0.477	−0.181	−0.319	0.149	−0.205
Coh	−0.415	0.251	0.670	−0.232	−0.064	−0.262
TCC	−0.393	0.349	0.753	−0.176	−0.014	−0.144
Eigen val.	2.885	2.205	2.020	1.586	1.220	1.013
% Var.	19.233	15.303	13.467	10.576	8.134	6.755
%Cum. Var.	19.233	34.536	48.003	58.579	66.713	**73.468**

Table 6
Principal components for Model III

Metrics	PC1	PC2	PC3	PC4	PC5	PC6
DIT	0.030	−0.544	0.355	−0.034	0.420	0.280
CLD	−0.317	0.398	−0.282	0.232	0.428	−0.086
NOC	−0.276	0.596	−0.302	0.208	0.464	0.103
CBO	0.463	−0.217	0.316	0.080	0.239	−0.467
RFC	0.740	0.061	0.217	0.699	0.016	0.076
AMMIC	0.267	−0.222	0.535	−0.190	0.354	0.385
OMMIC	0.569	0.024	0.118	0.752	−0.084	−0.109
DMMEC	−0.114	0.454	−0.058	0.135	0.513	0.182
OMMEC	0.166	0.203	−0.040	−0.242	−0.364	0.290
LCOM1	0.730	0.245	−0.190	0.159	−0.094	0.276
LCOM4	0.518	0.294	−0.272	−0.299	−0.123	0.344
CO	−0.098	0.530	0.392	0.044	−0.120	0.010
LCOM5	0.542	0.469	−0.181	−0.268	−0.050	−0.171
Coh	−0.352	0.374	0.630	0.044	−0.260	0.116
TCC	−0.307	0.497	0.712	0.064	−0.165	0.018
NA	0.218	0.330	0.263	−0.323	0.218	−0.517
NM	0.458	0.092	0.218	−0.502	0.366	0.134
Stmts	0.374	0.142	0.000	−0.390	−0.023	−0.275
Eigen Val.	3.095	2.359	2.066	1.843	1.478	1.196
% Var.	17.195	13.104	11.479	10.238	8.213	6.645
% Cum. Var.	17.195	30.299	41.778	52.016	60.229	**66.874**

Model III of discriminant system achieves 100% correctness in identifying fault prone classes, has a poor prediction of non-fault prone classes.

Completeness is calculated as the ratio of the total number of faults in the predicted fault prone classes to the total number of faults present in the actually fault prone classes (158) of the software. The number of faults identified by each of the prediction model is given in Table 12 (in braces). From the results, it is found that, Model III of PNN is able to identify the maximum of 157 out of 158 faults present in the classes. In general among the three models in each method, Model III prediction results are good. This indicates that the compound effect of the size and OO measures have a strong relationship to fault proneness. Among the models, PNN based models classify the fault prone classes with more faults very accurately.

The effectiveness and efficiency of the models are computed as defined in Section 2 and presented in the Tables 13 and 14, respectively. Both effectiveness and efficiency captures the productive effort to be spent in inspecting the real fault-prone classes. When we achieve 100% effectiveness, it means all efforts spent in inspection are fruitful. Otherwise the value of (100 − % effectiveness) gives the effort to be wasted in the inspection process. Neural network based models classify the faulty modules such that the waste of effort during inspection is very much minimal.

Efficiency computes the ratio of productive effort spent (for the real fault-prone classes) in the inspection process to the over all effort (for both fault prone and non-fault prone classes). Even though discriminant Model III gets 100% efficiency, it was due to the 90 actual fault prone classes predicted as the fault prone and none of the non-fault prone class is predicted as fault prone (refer Table 7).

Table 7
Results of discriminant analysis method

Group	Model I			Model II			Model III		
	Prediction model		Total	Prediction model		Total	Prediction model		Total
	G_1 fault	G_2 non-fault		G_1 fault	G_2 non-fault		G_1 fault	G_2 non-fault	
Actual									
G_1 fault	99	59	158	100	58	158	90	68	158
	62.7%	37.3%	100%	63.3%	36.7%	100%	56.96%	43%	100%
		Type II error			*Type II error*			*Type II error*	
G_2 non-fault	83	154	237	92	145	237	0	237	237
	35%	64.98%	100%	38.8%	61.18%	100%	0%	100%	100%
	Type I error			*Type I error*			*Type I error*		
Total	182	213	395	192	203	395	90	305	395
Percent	46.08%	53.9%	100%	48.6%	51.4%	100%	22.8%	77.2%	100%
	Overall misclassification rate: 35.95%			Overall misclassification rate: 37.97%			Overall misclassification rate: 17.2%		

Table 8
Results of logistic regression method

Group	Model I			Model II			Model III		
	Prediction model		Total	Prediction model		Total	Prediction model		Total
	G_1 fault	G_2 non-fault		G_1 fault	G_2 non-fault		G_1 fault	G_2 non-fault	
Actual									
G_1 fault	116	42	158	112	46	158	102	56	158
	73.4%	26.6%	100%	70.8%	29.1%	100%	64.6%	35.4%	100%
		Type II error			*Type II error*			*Type II error*	
G_2 non-fault	104	133	237	121	116	237	86	152	237
	43.9%	56.1%	100%	51.1%	48.9%	100%	36.1%	63.86%	100%
	Type I error			*Type I error*			*Type I error*		
Total	220	175	395	233	162	395	188	208	395
Percent	55.7%	44.3%	100%	58.9%	41.0%	100%	47.6%	56.7%	100%
	Overall misclassification rate: 36.9%			Overall misclassification rate: 42.3%			Overall misclassification rate: 35.9%		

Table 9
Results of back propagation neural networks

Group	Model I			Model II			Model III		
	Prediction model		Total	Prediction model		Total	Prediction model		Total
	G_1 fault	G_2 non-fault		G_1 fault	G_2 non-fault		G_1 fault	G_2 non-fault	
Actual									
G_1 fault	112	46	158	146	12	158	125	33	158
	70.8%	29.1%	100%	92.4%	7.6%	100%	79.1%	20.9%	100%
		Type II error			*Type II error*			*Type II error*	
G_2 non-fault	73	164	237	63	174	237	41	196	237
	30.8%	69.2%	100%	26.6%	73.4%	100%	17.3%	82.7%	100%
	Type I error			*Type I error*			*Type I error*		
Total	185	210	395	209	186	395	166	229	395
Percent	46.8%	53.3%	100%	52.9%	47.1%	100%	42%	58%	100%
	Overall misclassification rate: 30.1%			Overall misclassification rate: 18.9%			Overall misclassification rate:18.7%		

Table 10
Results of probabilistic neural networks

Group	Model I			Model II			Model III		
	Prediction model		Total	Prediction model		Total	Prediction model		Total
	G_1 fault	G_2 non-fault		G_1 fault	G_2 non-fault		G_1 fault	G_2 non-fault	
Actual									
G_1 fault	153	3	158	155	3	158	157	1	158
	96.8%	1.89%	100%	98.1%	1.9%	100%	99.3%	0.63%	100%
		Type II error			*Type II error*			*Type II error*	
G_2 non-fault	7	232	237	20	217	237	2	235	237
	2.95%	97.9%	100%	8.4%	91.6%	100%	0.84%	99.2%	100%
	Type I error			*Type I error*			*Type I error*		
Total	160	235	395	175	220	395	159	236	395
Percent	40.5%	59.5%	100%	44.3%	55.7%	100%	40.2%	59.7%	100%
	Overall misclassification rate: 2.53%			Overall misclassification rate: 5.8%			Overall misclassification rate: 0.76%		

However, it has 56 fault-prone classes predicted as not fault prone (35.4% *Type II error*). PNN has minimal *Type I & Type II error* and able to achieve 98.1% efficiency.

From the results it is found that among the four methods, PNN based prediction models perform well in all aspects. Among the three models, Model III based prediction accuracy is high. It shows that inclusion of size measures in prediction gives better results compared to the results of using OO metrics alone.

5.1. Threats for validity

Although this investigation is based on a large code sample, there are a number of threats to its validity. The investigation is carried out over the small system (having 10,000–15,000 lines of code) developed by developers who are not well-trained and experienced as professional programmers. The problem for which software solution developed was having a limited conceptual complexity.

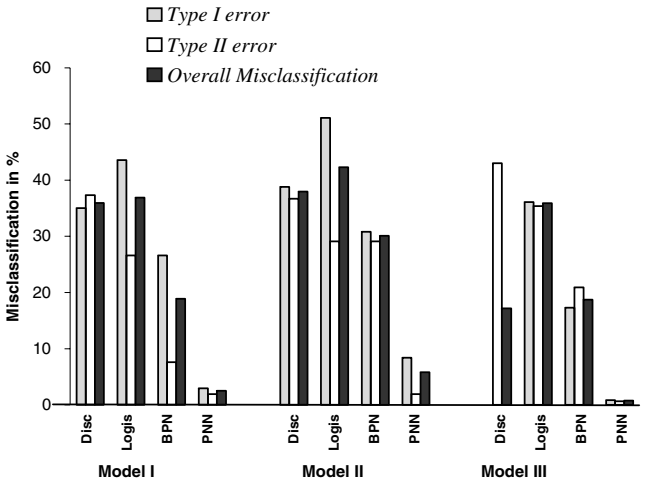

Fig. 1. Consolidated results of misclassification rates.

Table 11
Results of correctness (in %)

Method	Model I	Model II	Model III
PNN	95.63	88.6	98.74
BPN	69.9	60.54	75.3
Disc.	54.4	52	100
Logistic	52.7	48.07	54.3

Table 12
Results of completeness (in %)

Method	Model I	Model II	Model III
PNN	93.16 (147)	98.71 (156)	99.92 (157)
BPN	71.36 (113)	73.45 (116)	76.74 (121)
Disc.	61.32 (97)	67.52 (107)	62.51 (99)
Logistic	60.25 (95)	62.63 (99)	63.5 (100)

Table 13
Results of effectiveness (in %)

Method	Model I	Model II	Model III
PNN	96.8	98.1	99.3
BPN	92.4	70.8	79.1
Disc.	62.7	63.3	56.96
Logistic	73.4	70.9	64.6

Table 14
Results of efficiency (in %)

Method	Model I	Model II	Model III
PNN	94	81.25	98.1
BPN	60.66	50.25	68
Disc.	45.5	42	100
Logistic	42.64	38	44.06

From the experiment it has been demonstrated that the measures have relationship with fault proneness through statistical and neural network models. But this relationship does not demonstrate casual relationship only controlled experiments, where the measure would be varied in a controlled manner and all other factors would be held constant could really demonstrate the causality.

6. Related work

The limitations in using non-Object-Oriented metrics for Object-Oriented environment were realized in 1990's. Chidamber and Kemerer [8] introduced a metric suite with six exclusive measures for Object-Oriented environment. Following them many introduced new measures for Object-Oriented environment. Li and Henry [18] introduced a set of class measures for system maintenance. Lorenz and Kidd [19] proposed measures for class size, inheritance, internals and externals. Abreu et al. [1] proposed a set of system level measures. Briand et al. [5] introduced an extended set of coupling measures. Benlarbi et al. [3] proposed a set of polymorphism measures.

As there were several measures found for each of the Object-Oriented features, many researchers attempted on the validation studies to find their application in quality prediction. Some of them worked on theoretical validation to prove the relationship but most of the studies were carried out on an empirical data set. Such studies are experimented by constructing prediction models for the quality attribute representatives such as fault proneness [3,6,10,12], productivity [9], effort [2], maintenance modifications [18] and fault count [24].

Even though many techniques for developing multivariate models were present like decision trees, neural networks, discriminant analysis, optimized set reduction, rough set analysis, linear and logistic regression, only few techniques were attempted in quality prediction [2,3,6,9,10,12,18]. Many approaches for evaluating the quality of the prediction models are available for example, statistical significance of co-efficient [10], goodness of fit [7], cross validation [8], data splitting [24] etc.

Even though Khoshgoftaar et al. [17] used Back Propagation Neural network to predict fault proneness for the software modules developed in procedural paradigm, Tong-Seng et al. [24] first attempted neural network based prediction for OO environment. They applied General Regression Neural Networks to empirically validate nine Object-Oriented metrics to predict the value of fault count.

This work distinguishes itself from others in the following ways:

- In this work, Probabilistic Neural Network and BPN are used for the fault proneness classification.
- Most of the earlier studies used the empirical data from industry but we attempted on the data generated in an academic institution.
- Five different quality parameters are used to evaluate the prediction models, which capture the various aspects of the model quality.
- All measures (a total of 64) considered in Briand et al. [6] are used and by a stepwise process it was reduced to 18.

7. Conclusion

In the development of prediction models to classify the fault prone modules, more reliable approaches are expected by the software developers to reduce the misclassifications. In this paper, two neural network based approaches, which perform better than the statistical approaches are introduced. Among the two, PNN was highly robust in nature which was studied through the five quality parameters. This work did not account for the severity of faults. When the models need to be developed in organizations, the severity is required to be assigned. However, the newly introduced models can be used to identify fault prone classes that may cause any type of failure.

Appendix A

SPSS (Statistical Package for Social Sciences) is one of the most widely available and powerful statistical software packages. It covers a broad range of statistical procedures that allows us to summarize data (e.g., Compute mean, standard deviation,...), determine whether there are significant differences between groups (e.g., *t*-tests, analysis of variance), examine relationships among variables (e.g., correlation, multiple regression) and graph results (e.g., bar charts, line graphs).

The Neural Network toolbox extends MATLAB with tools for designing, implementing, visualizing and simulating neural networks. The toolbox features 15 neural models, five learning algorithms and host of useful utilities integrated in an easy to use interface. The modular, open and extensible design of the toolbox simplifies the creation of customized functions and networks.

References

[1] Abreu, F.B, Melo W. Evaluating the impact of object-oriented design on software quality, in: Proceedings of the Third International Software Metrics Symposium, 1996, pp. 90–99.
[2] V. Basili, L. Briand, W. Melo, A validation of object-oriented design metrics as quality indicators, IEEE Transactions on Software Engineering 22 (10) (1996) 751–761.
[3] Benlarbi, S, Melo, W, Polymorphism measures for early risk prediction, in: Proceedings of the 21st International Conference on Software Engineering, 1999, pp. 334–344.
[4] L. Briand, J. Daly, J. Wust, A unified framework for cohesion measurement in object-oriented systems, Empirical Software Engineering Journal 3 (1) (1998) 65–117.
[5] L. Briand, J. Daly, J. Wust, A unified framework for coupling measurement in object-oriented systems, IEEE Transactions on Software Engineering 25 (1) (1999) 91–121.
[6] L. Briand, J. Wust, J. Daly, D. Victor Poter, Exploring the relationships between design measures and software quality in object-oriented systems, Journal of Systems and Software 51 (2000) 245–273.
[7] L. Briand, J. Wust, Empirical studies of quality models in object-oriented systems, in: Marvin Zelkowitz (Ed.), Advances in Computers, vol. 56, Academic Press, 2002, pp. 1–44.
[8] S. Chidamber, C. Kemerer, A metrics suite for object-oriented design, IEEE Transactions on Software Engineering 20 (6) (1994) 476–493.
[9] S. Chidamber, D. Darcy, C. Kemerer, Managerial use of metrics for object-oriented software an exploratory analysis, IEEE Transactions on Software Engineering 24 (8) (1998) 629–639.
[10] G. Denaro, M. Pezze, An empirical evaluation of fault proneness models, in: Proceedings of International Conference on Software Engineering (ICSE), Argentina, May 19–25, 2002, pp. 241–251.
[11] K.E. Emam, W. Melo, J.C. Machado, The prediction of faulty classes using object-oriented design metrics, Journal of Systems and Software 56 (2001) 63–75.
[12] F. Fioravanti, P. Nesi, A study on fault-proneness detection of object-oriented systems. Proceedings of Fifth European Conference on Software Maintenance and Reengineering, Portugal, March 14–16, 2001, pp. 121–130.
[13] B. Henderson-Sellers, Object-Oriented Metrics: Measures of complexity, Prentice Hall, Englewood Cliffs, NJ, 1996.
[14] R. Hochman, T.M. Khoshgoftaar, E.B. Allen, J.P. Hudepohl, Evolutionary neural networks: a robust approach to software reliability problems. Proceedings of the Eighth International Symposium on Software Reliability Engineering, USA, November, 1997, pp. 13–26.
[15] Howard Demuth, Mark Beale, Neural Network Toolbox: For use with MATLAB. The Mathworks Inc., Natick Massachusets, USA, 1998.
[16] T.M. Khoshgoftaar, R.M. Szabo, P.J. Guasti, Exploring the behaviour of neural network software quality models, Software Engineering Journal 1 (1995) 89–96.
[17] T.M. Khoshgoftaar, E.B. Allen, J.P. Hudepohl, S.J. Aud, Application of neural networks to software quality modeling of a very large telecommunications systems, IEEE Transactions on Neural Networks 8 (4) (1997) 902–909.
[18] W. Li, S. Henry, Object-oriented metrics that predict maintainability, Journal of Systems and Software 23 (1993) 111–122.
[19] M. Lorenz, J. Kidd, Object-Oriented Software Metrics, Prentice Hall, Englewood cliffs, NJ, 1994.
[20] M.C. Ohisson, P. Runeson, Experience from replicating empirical studies on prediction models. Proceedings of Eighth IEEE Software Metrics Meeting Symposium, Canada, April, 2002, pp. 217–226.
[21] D.F. Specht, Probabilistic neural networks, Neural Networks 3 (1990) 109–118.
[22] G.B. Tabachnick, S.F. Linda, Using Multivariate Statistics, fourth ed., Boston, 2001.
[23] M.H. Tang, M.H. Kao, M.H. Chen, An empirical study on object-oriented metrics. Proceedings of Sixth International Conference on Software Metrics Symposium, 1999, pp. 242–249.
[24] Tong-Seng Quah, Mie Mie Thet Thwin, Application of neural networks for software quality prediction using OO metrics, Proceedings of IEEE International Conference on Software Maintenance, 2003, pp. 106–1073.
[25] X. Yuan, T.M. Khoshgoftaar, E.B. Allen, K. Ganesan, An application of fuzzy clustering to software quality prediction. Proceedings of 3rd IEEE Symposium on ASSET'00, March 24–25, 2000, pp. 85–91.