Hindawi Publishing Corporation
ISRN Software Engineering
Volume 2014, Article ID 251083, 15 pages
http://dx.doi.org/10.1155/2014/251083

Research Article

Statistical and Machine Learning Methods for Software Fault Prediction Using CK Metric Suite: A Comparative Analysis

Yeresime Suresh, Lov Kumar, and Santanu Ku. Rath

Department of Computer Science and Engineering, National Institute of Technology, Rourkela, Odisha 769008, India

Correspondence should be addressed to Yeresime Suresh; suresh.vec04@gmail.com

Received 31 August 2013; Accepted 16 January 2014; Published 4 March 2014

Academic Editors: K. Framling, Z. Shen, and S. K. Shukla

Experimental validation of software metrics in fault prediction for object-oriented methods using statistical and machine learning methods is necessary. By the process of validation the quality of software product in a software organization is ensured. Object-oriented metrics play a crucial role in predicting faults. This paper examines the application of linear regression, logistic regression, and artificial neural network methods for software fault prediction using Chidamber and Kemerer (CK) metrics. Here, fault is considered as dependent variable and CK metric suite as independent variables. Statistical methods such as linear regression, logistic regression, and machine learning methods such as neural network (and its different forms) are being applied for detecting faults associated with the classes. The comparison approach was applied for a case study, that is, Apache integration framework (AIF) version 1.6. The analysis highlights the significance of weighted method per class (WMC) metric for fault classification, and also the analysis shows that the hybrid approach of radial basis function network obtained better fault prediction rate when compared with other three neural network models.

1. Introduction

Present day software development is mostly based on object-oriented paradigm. The quality of object-oriented software can be best assessed by the use of software metrics. A number of metrics have been proposed by researchers and practitioners to evaluate the quality of software. These metrics help to verify the quality attributes of a software such as effort and fault proneness.

The usefulness of these metrics lies in their ability to predict the reliability of the developed software. In practice, software quality mainly refers to reliability, maintainability, and understandability. Reliability is generally measured by the number of faults found in the developed software. Software fault prediction is a challenging task for researchers before the software is released. Hence, accurate fault prediction is one of the major goals so as to release a software having the least possible faults.

This paper aims to assess the influence of CK metrics, keeping in view of predicting faults for an open-source software product. Statistical methods such as linear regression and logistic regression are used for classification of faulty classes. Machine learning algorithms such as artificial neural network (ANN), functional link artificial neural network (FLANN), and radial basis function network (RBFN) are applied for prediction of faults, and probabilistic neural network (PNN) is used for classification of faults. It is observed in literature that metric suites have been validated for small data sets. In this approach, the results achieved for an input data set of 965 classes were validated by comparing with the results obtained by Basili et al. [1] for statistical analysis.

The rest of the paper is organized as follows. Section 2 summarizes software metrics and their usage in fault prediction. Section 3 highlights research background. Section 4 describes the proposed work for fault prediction by applying various statistical and machine learning methods. Section 5 highlights the parameters used for evaluating the performance of each of the applied techniques. Section 6 presents the results and analysis of fault prediction. Section 7 concludes the paper with scope for future work.

2. Related Work

This section presents a review of the literature on the use of software metrics and their application in fault prediction. The most commonly used metric suites, indicating the quality of any software, are McCabe [2], Halstead [3], Li and Henry [4], CK metric [5], Abreu MOOD metric suite [6], Lorenz and Kidd [7], Martin's metric suite [8], Tegarden et al. [9], Melo and Abreu [10], Briand et al. [11], Etzkorn et al. [12], and so forth. Out of these metrics, CK metric suite is observed to be used very often by the following authors as mentioned in Table 1 for predicting faults at class level.

Basili et al. [1] experimentally analyzed the impact of CK metric suite on fault prediction. Briand et al. [13] found out the relationship between fault and the metrics using univariate and multivariate logistic regression models. Tang et al. [14] investigated the dependency between CK metric suite and the object-oriented system faults. Emam et al. [15] conducted empirical validation on Java application and found that export coupling has great influence on faults. Khoshgoftaar et al. [16, 17] conducted experimental analysis on telecommunication model and found that ANN model is more accurate than any discriminant model. In their approach, nine software metrics were used for modules developed in procedural paradigm. Since then, ANN approach has taken a rise in their usage for prediction modeling.

Table 1: Fault prediction using CK metrics.

Author	Prediction technique
Basili et al. [1]	Multivariate logistic regression
Briand et al. [13]	Multivariate logistic regression
Kanmani and Rymend [29]	Regression, neural network
Nagappan and Laurie [30]	Multiple linear regression
Olague et al. [31]	Multivariate logistic regression
Aggarwal et al. [32]	Statistical regression analysis
Wu [33]	Decision tree analysis
Kapila and Singh [34]	Bayesian inference

3. Research Background

The following subsections highlight the data set being used for fault prediction. Data are normalized to obtain better accuracy, and then dependent and independent variables are chosen for fault prediction.

3.1. Empirical Data Collection. Metric suites are used and defined for different goals such as fault prediction, effort estimation, reusability, and maintenance. In this paper, the most commonly used metric, that is, CK metric suite, [5] is used for fault prediction.

The CK metric suite consists of six metrics, namely, weighted method per class (WMC), depth of inheritance tree (DIT), number of children (NOC), coupling between objects (CBO), response for class (RFC), and lack of cohesion (LCOM) [5]. Table 2 gives a short note on the six CK metrics and the threshold for each of the six metrics.

The metric values of the suite are extracted using Chidamber and Kemerer Java Metrics (CKJM) tool. CKJM tools extract object-oriented metrics by processing the byte code of compiled Java classes. This tool is being used to extract metric values for three versions of Apache integration framework (AIF, an open-source framework) available in the Promise data repository [18]. The versions of the AIF used from the repository are developed in Java language. The CK metric values of the AIF are used for fault prediction.

3.2. Data Normalization. ANN models accept normalized data which lie in the range of 0 to 1. In the literature it is observed that techniques such as Min-Max normalization, Z-Score normalization, and Decimal scaling are being used for normalizing the data. In this paper, Min-Max normalization [19] technique is used to normalize the data.

Min-Max normalization performs a linear transformation on the original data. Each of the actual data d of attribute p is mapped to a normalized value d' which lies in the range of 0 to 1. The Min-Max normalization is calculated by using the equation:

$$\text{Normalized}\,(d) = d' = \frac{d - \min(P)}{\max(p) - \min(p)}, \quad (1)$$

where $\min(p)$ and $\max(p)$ represent the minimum and maximum values of the attribute, respectively.

3.3. Dependent and Independent Variables. The goal of this study is to explore the relationship between object-oriented metrics and fault proneness at the class level. In this paper, a fault in a class is considered as a dependent variable and each of the CK metrics is an independent variable. It is intended to develop a function between fault of a class and CK metrics (WMC, DIT, NOC, CBO, RFC, and LCOM). Fault is a function of WMC, DIT, NOC, CBO, RFC, and LCOM and can be represented as shown in the following equation:

$$\text{Faults} = f\,(\text{WMC}, \text{DIT}, \text{NOC}, \text{CBO}, \text{RFC}, \text{LCOM})\,. \quad (2)$$

4. Proposed Work for Fault Prediction

The following subsections highlight the various statistical and machine learning methods used for fault classification.

4.1. Statistical Methods. This section describes the application of statistical methods for fault prediction. Regression analysis methods such as linear regression and logistic regression analysis are applied. In regression analysis, the value of unknown variable is predicted based on the value of one or more known variables.

4.1.1. Linear Regression Analysis. Linear regression is a statistical technique and establishes a linear (i.e., straight-line) relationship between variables. This technique is used when faults are distributed over a wide range of classes.

Table 2: CK metric suite.

CK metric	Description	Value
WMC	Sum of the complexities of all class methods	Low
DIT	Maximum length from the node to the root of the tree	<six
NOC	Number of immediate subclasses subordinate to a class in the class hierarchy	Low
CBO	Count of the number of other classes to which it is coupled	Low
RFC	A set of methods that can potentially be executed in response to a message received by an object of that class	Low
LCOM	Measures the dissimilarity of methods in a class via instanced variables	Low

Linear regression analysis is of two types:

(a) univariate linear regression, and

(b) multivariate linear regression.

Univariate linear regression is based on

$$Y = \beta_0 + \beta_1 X, \tag{3}$$

where Y represents dependent variables (accuracy rate for this case) and X represents independent variables (CK metrics for this case).

In case of multivariate linear regression, the linear regression is based on

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \cdots + \beta_p X_p, \tag{4}$$

where X_i is the independent variable, β_0 is a constant, and y is the dependent variable. Table 8 shows the result of linear regression analysis for three versions of AIF.

4.1.2. Logistic Regression Analysis. Logistic regression analysis is used for predicting the outcome of dependent variables based on one or more independent variable(s). A dependent variable can take only two values. So the dependent variable of a class containing bugs is divided into two groups, one group containing zero bugs and the other group having at least one bug.

Logistic regression analysis is of two types:

(a) univariate logistic regression, and

(b) multivariate logistic regression.

(a) Univariate Logistic Regression Analysis. Univariate logistic regression is carried out to find the impact of an individual metric on predicting the faults of a class. The univariate logistic regression is based on

$$\pi(x) = \frac{e^{\beta_0 + \beta_1 X_1}}{1 + e^{\beta_0 + \beta_1 X_1}}, \tag{5}$$

where x is an independent variable and β_0 and β_1 represent the constant and coefficient values, respectively. Logit function can be developed as follows:

$$\text{logit}\,[\pi(x)] = \beta_0 + \beta_1 X, \tag{6}$$

where π represents the probability of a fault found in the class during validation phase.

The results of univariate logistic regression for AIF are tabulated in Table 9. The values of obtained coefficient are the estimated regression coefficients. The probability of faults being detected for a class is dependent on the coefficient value (positive or negative). Higher coefficient value means greater probability of a fault being detected. The significance of coefficient value is determined by the P value. The P value was assessed based on the significance level (α). R coefficient is the proportion of the total variation in the dependent variable explained in the regression model. High value of R indicates greater correlation between faults and the CK metrics.

(b) Multivariate Logistic Regression Analysis. Multivariate logistic regression is used to construct a prediction model for the fault proneness of classes. In this method, metrics are used in combination. The multivariate logistic regression model is based on the following equation:

$$\pi(x) = \frac{e^{\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \cdots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \cdots + \beta_p X_p}}, \tag{7}$$

where x_i is the independent variable, π represents the probability of a fault found in the class during validation phase, and p represents the number of independent variables. The Logit function can be formed as follows:

$$\text{logit}\,[\pi(x)] = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \cdots + \beta_p X_p. \tag{8}$$

Equation (8) shows that logistic regression is really just a standard linear regression model, where the dichotomous outcome of the result is transformed by the logit transform. The value of $\pi(x)$ lies in the range $0 > \pi(x) < 1$. After the logit transforms the value of $\pi(x)$ lies in the range $-\infty > \pi(x) < +\infty$.

4.2. Machine Learning Methods. Besides the statistical approach, this paper also implements four other machine learning techniques. Machine learning techniques have been used in this paper to predict the accuracy rate in fault prediction using CK metric suite.

This section gives a brief description of the basic structure and working of machine learning methods applied for fault prediction.

4.2.1. Artificial Neural Network. Figure 1 shows the architecture of ANN, which contains three layers, namely, input layer, hidden layer, and output layer. Computational features involved in ANN architecture can be very well applied for fault prediction.

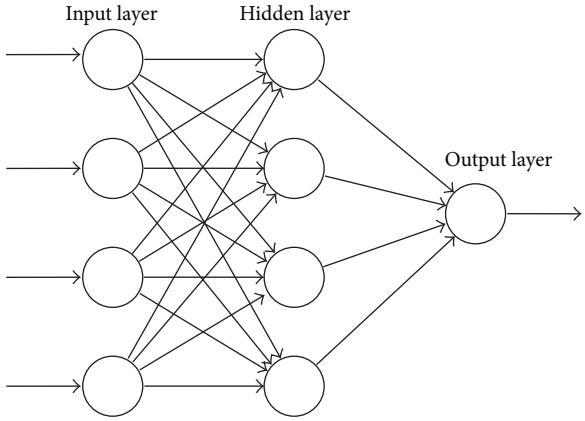

Figure 1: A typical FFNN.

In this paper for input layer, linear activation function has been used; that is, the output of the input layer "O_i" is input of the input layer "I_i," which is represented as follows:

$$O_i = I_i. \tag{9}$$

For hidden layer and output layer, sigmoidal (squashed-S) function is used. The output of hidden layer O_h for input of hidden layer I_h is represented as follows:

$$O_h = \frac{1}{1 + e^{-I_h}}. \tag{10}$$

Output of the output layer "O_o" for the input of the output layer "O_i" is represented as follows:

$$O_o = \frac{1}{1 + e^{-O_i}}. \tag{11}$$

A neural network can be represented as follows:

$$Y' = f(W, X), \tag{12}$$

where X is the input vector, Y' is the output vector, and W is weight vector. The weight vector W is updated in every iteration so as to reduce the mean square error (MSE) value. MSE is formulated as follows:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i' - y_i\right)^2, \tag{13}$$

where y is the actual output and y' is the expected output. In the literature, different methods are available to update weight vector ("W") such as Gradient descent method, Newton's method, Quasi-Newton method, Gauss Newton Conjugate-gradient method, and Levenberg Marquardt method. In this paper, Gradient descent and Levenberg Marquardt methods are used for updating the weight vector W.

(a) Gradient Descent Method. Gradient descent is one of the methods for updating the weight during learning phase [20]. Gradient descent method uses first-order derivative of total error to find the *minima* in error space. Normally gradient vector G is defined as the first-order derivative of error function. Error function is represented as follows:

$$E_k = \frac{1}{2}(T_k - O_k)^2 \tag{14}$$

and G is given as:

$$G = \frac{\partial d}{\partial dW}(E_k) = \frac{\partial d}{\partial dW}\left(\frac{1}{2}(T_k - O_k)^2\right). \tag{15}$$

After computing the value of gradient vector G in each iteration, weighted vector W is updated as follows:

$$W_{k+1} = W_k - \alpha G_k, \tag{16}$$

where W_{k+1} is the updated weight, W_k is the current weight, G_k is a gradient vector, and α is the learning parameter.

(b) Levenberg Marquardt (LM) Method. LM method locates the minimum of multivariate function in an iterative manner. It is expressed as the sum of squares of nonlinear real-valued functions [21, 22]. This method is used for updating the weights during learning phase. LM method is fast and stable in terms of its execution when compared with gradient descent method (LM method is a combination of steepest descent and Gauss-Newton methods). In LM method, weight vector W is updated as follows:

$$W_{k+1} = W_k - \left(J_k^T J_k + \mu I\right)^{-1} J_k e_k, \tag{17}$$

where W_{k+1} is the updated weight, W_k is the current weight, J is Jacobian matrix, and μ is combination coefficient; that is, when μ is very small then it acts as Gauss-Newton method and if μ is very large then it acts as Gradient descent method.

Jacobian matrix is calculated as follows:

$$J = \begin{bmatrix} \frac{\partial d}{\partial dW_1}(E_{1,1}) & \frac{\partial d}{\partial dW_2}(E_{1,1}) & \cdots & \frac{\partial d}{dW_N}(E_{1,1}) \\ \frac{\partial d}{\partial dW_1}(E_{1,2}) & \frac{\partial d}{\partial dW_2}(E_{1,2}) & \cdots & \frac{\partial d}{\partial dW_N}(E_{1,2}) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial d}{\partial dW_1}(E_{P,M}) & \frac{\partial d}{\partial dW_2}(E_{P,M}) & \cdots & \frac{\partial d}{\partial dW_N}(E_{P,M}) \end{bmatrix}, \tag{18}$$

where N is number of weights, P is the number of input patterns, and M is the number of output patterns.

4.2.2. Functional Link Artificial Neural Network (FLANN). FLANN, initially proposed by Pao [23], is a flat network having a single layer; that is, the hidden layers are omitted. Input variables generated by linear links of neural network are linearly weighed. Functional links act on elements of input variables by generating a set of linearly independent functions. These links are evaluated as functions with the variables as the arguments. Figure 2 shows the single layered

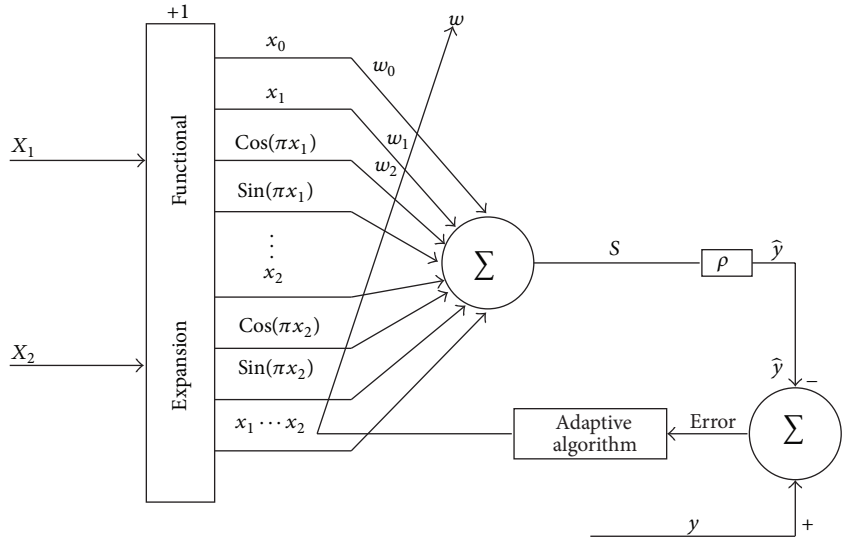

Figure 2: Flat net structure of FLANN.

architecture of FLANN. FLANN architecture offers less computational overhead and higher convergence speed when compared with other ANN techniques.

Using FLANN, output is calculated as follows:

$$\hat{y} = \sum_{i=1}^{n} W_i X_i, \tag{19}$$

where $\hat{y}$ is the predicted value, W is the weight vector, and X is the functional block, and is defined as follows:

$$X = \left[1, x_1, \sin(\pi x_1), \cos(\pi x_1), x_2, \sin(\pi x_2), \cos(\pi x_2), \ldots\right] \tag{20}$$

and weight is updated as follows:

$$W_i(k+1) = W_i(k) + \alpha e_i(k) x_i(k) \tag{21}$$

having α as the learning rate and e_i as the error value. "e_i" is formulated as follows:

$$e_i = y_i - \hat{y}_i, \tag{22}$$

here y and $\hat{y}$ represent actual and the obtained (predicted) values, respectively.

4.2.3. Radial Basis Function Network (RBFN). RBFN is a feed-forward neural network (FFNN), trained using supervised training algorithm. RBFN is generally configured by a single hidden layer, where the activation function is chosen from a class of functions called basis functions.

RBFN is one of the ANN techniques which contains three layers, namely, input, hidden, and output layer. Figure 3 shows the structure of a typical RBFN in its basic form involving three entirely different layers. RBFN contains h number of hidden centers represented as $C_1, C_2, C_3, \ldots, C_h$.

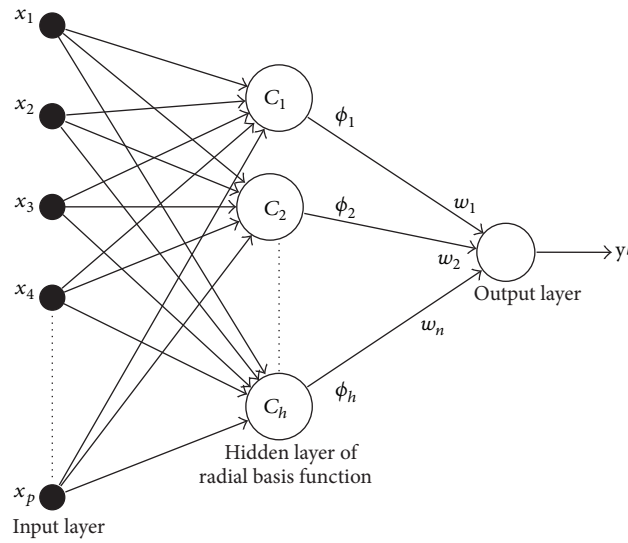

Figure 3: RBFN network.

The target output is computed as follows:

$$y' = \sum_{i=1}^{n} \phi_i W_i, \tag{23}$$

where W_i is the weight of the ith center, ϕ is the radial function, and y' is the target output. Table 3 shows the various radial functions available in the literature.

In this paper, Gaussian function is used as a radial function, and z the distance vector is calculated as follows:

$$z = \left\| x_j - c_j \right\|, \tag{24}$$

where x_j is input vector that lies in the receptive field for center c_j. In this paper, gradient descent learning and hybrid learning techniques are used for updating weight and center, respectively.

Table 3: Radial function.

Radial function	Mathematical expression
Gaussian radial function	$\phi(z) = e^{-(z^2/2\sigma^2)}$
Thin plate spline	$\phi(z) = z^2 \log z$
Quadratic	$\phi(z) = (z^2 + r^2)^{1/2}$
Inverse quadratic	$\phi(z) = \dfrac{1}{(z^2 + r^2)^{1/2}}$

The advantage of using RBFN lies in its training rate which is faster when compared with propagation networks and is less susceptible to problem with nonstationary inputs.

(a) Gradient Descent Learning Technique. Gradient descent learning is a technique used for updating the weight W and center C. The center C in gradient learning is updated as:

$$C_{ij}(k+1) = C_{ij}(k) - \eta_1 \frac{\partial d}{\partial d C_{ij}}(E_k) \tag{25}$$

and weight W is updated as:

$$W_i(k+1) = W_i(k) - \eta_2 \frac{\partial d}{\partial d W_i}(E_k), \tag{26}$$

where η_1 and η_2 are the learning coefficients for updating center and weight, respectively.

(b) Hybrid Learning Technique. In hybrid learning technique, radial function relocates their center in self-organized manner while the weights are updated using learning algorithm. In this paper, least mean square (LMS) algorithm is used for updating the weights while the center is updated only when it satisfies the following conditions:

(a) Euclidean distance between the input pattern and the nearest center is greater than the threshold value, and

(b) MSE is greater than the desired accuracy.

After satisfying the above conditions, the Euclidean distance is used to find the centers close to x and then the centers are updated as follows:

$$C_i(k+1) = C_i(k) + \alpha\left(x - C_i(k)\right). \tag{27}$$

After every updation, the center moves closer to x.

4.2.4. Probabilistic Neural Network (PNN). PNN was introduced by Specht [24]. It is a feed-forward neural network, which has been basically derived from Bayesian network and statistical algorithm.

In PNN, the network is organized as multilayered feed-forward network with four layers such as input, hidden, summation, and output layer. Figure 4 shows the basic architecture of PNN.

The input layer first computes the distance from input vector to the training input vectors. The second layer consists of a Gaussian function which is formed using the given set of data points as centers. The summation layers sum up the contribution of each class of input and produce a net output which is vector of probabilities. The fourth layer determines the fault prediction rate.

Table 4: Confusion matrix to classify a class as faulty and not-faulty.

	No (prediction)	Yes (prediction)
No (actual)	True negative (TN)	False positive (FP)
Yes (actual)	False negative (FN)	True positive (TP)

PNN technique is faster when compared to multilayer perceptron networks and also is more accurate. The major concern lies in finding an accurate smoothing parameter "σ" to obtain better classification. The following function is used in hidden layer:

$$\phi(z) = e^{-(z^2/\sigma^2)}, \tag{28}$$

where $z = \|x - c\|$,

x is the input,

c is the center, and

z is the Euclidean distance between the center and the input vector.

5. Performance Evaluation Parameters

The following subsections give the basic definitions of the performance parameters used in statistical and machine learning methods for fault prediction.

5.1. Statistical Analysis. The performance parameters for statistical analysis can be determined based on the confusion matrix [25] as shown in Table 4.

5.1.1. Precision. It is defined as the degree to which the repeated measurements under unchanged conditions show the same results:

$$\text{Precision} = \frac{\text{TP}}{\text{FP} + \text{TP}}. \tag{29}$$

5.1.2. Correctness. Correctness as defined by Briand et al. [13] is the ratio of the number of modules correctly classified as fault prone to the total number of modules classified as fault prone:

$$\text{Correctness} = \frac{\text{TP}}{\text{FP} + \text{TP}}. \tag{30}$$

5.1.3. Completeness. According to Briand et al. [13], completeness is the ratio of number of faults in classes classified as fault prone to the total number of faults in the system:

$$\text{Completeness} = \frac{\text{TP}}{\text{FN} + \text{TP}}. \tag{31}$$

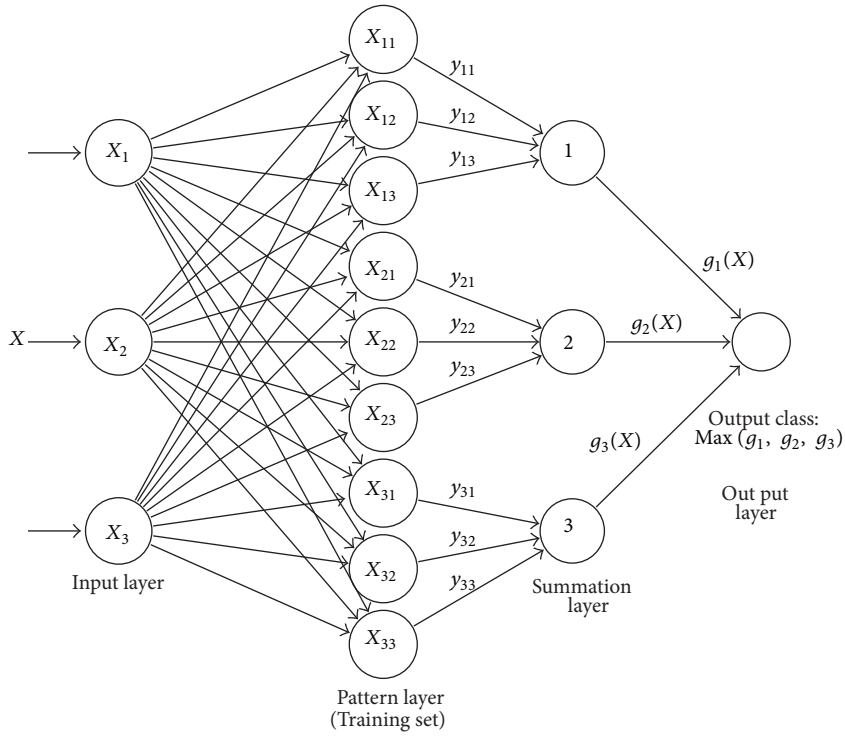

Figure 4: Basic structure of PNN.

5.1.4. Accuracy. Accuracy as defined by Yaun et al. [26] is the proportion of predicted fault prone modules that are inspected out of all modules:

$$\text{Accuracy} = \frac{\text{TN} + \text{TP}}{\text{TN} + \text{FP} + \text{FN} + \text{TP}}. \tag{32}$$

5.1.5. R^2 Statistic. R^2, also known as coefficient of multiple determination, is a measure of power of correlation between predicted and actual number of faults [25]. The higher the value of this statistic the more is the accuracy of the predicted model

$$R^2 = 1 - \frac{\sum_{i=1}^{n} \left(y_i - \widehat{y}_i\right)^2}{\sum_{i=1}^{n} \left(y_i - \overline{y}\right)^2}, \tag{33}$$

where y_i is the actual number of faults, $\widehat{y}_i$ is the predicted number of faults, and $\overline{y}$ is the average number of faults.

5.2. Machine Learning. Fault prediction accuracy for four of the applied ANN is determined by using performance evaluation parameters such as mean absolute error (MAE), mean absolute relative error (MARE), root mean square error (RMSE), and standard error of the mean (SEM).

5.2.1. Mean Absolute Error (MAE). This performance parameter determines how close the values of predicted and actual fault (accuracy) rate differ:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} \left| y_i - y_i' \right|. \tag{34}$$

5.2.2. Mean Absolute Relative Error (MARE). Consider

$$\text{MARE} = \frac{1}{n} \sum_{i=1}^{n} \frac{\left| y_i - y_i' \right|}{y_i}. \tag{35}$$

In (35), a numerical value of 0.05 is added in the denominator in order to avoid numerical overflow (division by zero). The modified MARE is formulated as:

$$\text{MARE} = \frac{1}{n} \sum_{i=1}^{n} \frac{\left| y_i - y_i' \right|}{y_i + 0.05}. \tag{36}$$

5.2.3. Root Mean Square Error (RMSE). This performance parameter determines the differences in the values of predicted and actual fault (accuracy) rate:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(y_i - y_i'\right)^2}. \tag{37}$$

In (35), (36), and (37), y_i is actual value and y_i' is expected value.

5.2.4. Standard Error of the Mean (SEM). It is the deviation of predicted value from the actual fault (accuracy) rate:

$$\text{SEM} = \frac{\text{SD}}{\sqrt{n}}, \tag{38}$$

where SD is sample standard deviation and "n" is the number of samples.

Table 5: Distribution of bugs for AIF version 1.6.

Number of classes	Percentage of bugs	Number of associated bugs
777	80.5181	0
101	10.4663	1
32	3.3161	2
16	1.6580	3
14	1.4508	4
6	0.6218	5
2	0.2073	6
3	0.3109	7
5	0.5181	8
1	0.1036	9
1	0.1036	10
3	0.3109	11
1	0.1036	13
1	0.1036	17
1	0.1036	18
1	0.1036	28
965	100.00	142

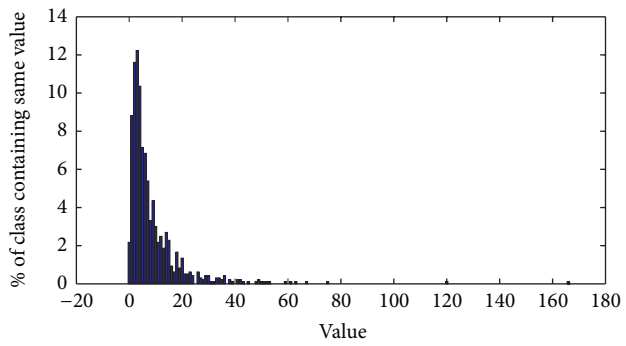

Figure 5: WMC of AIF version 1.6.

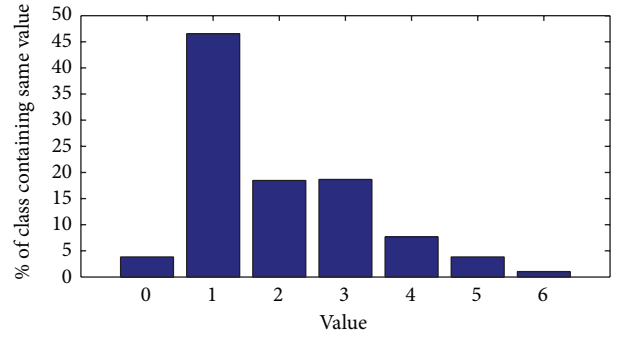

Figure 6: DIT of AIF version 1.6.

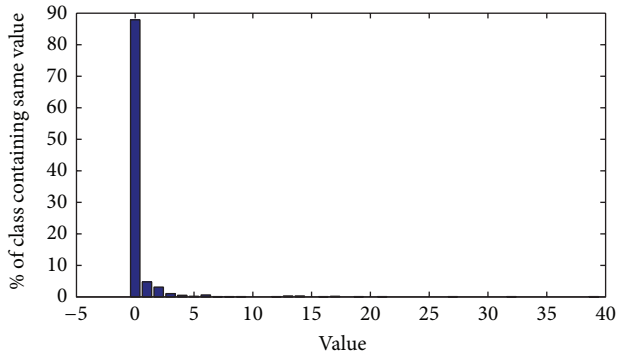

Figure 7: NOC of AIF version 1.6.

6. Results and Analysis

In this section, the relationship between value of metrics and the fault found in a class is determined. In this approach, the comparative study involves using six CK metrics as input nodes and the output is the achieved fault prediction rate. Fault prediction is performed for AIF version 1.6.

6.1. Fault Data. To perform statistical analysis, bugs were collected from Promise data repository [18]. Table 5 shows the distribution of bugs based on the number of occurrences (in terms of percentage of class containing number of bugs) for AIF version 1.6.

AIF version 1.6 contains 965 numbers of classes in which 777 classes contain zero bugs (80.5181%), 10.4663% of classes contain at least one bug, 3.3161% of classes contain a minimum of two bugs, 1.6580% of classes contain three bugs, 1.4508% of classes contain four bugs, 0.6218% of classes contain five bugs, 0.2073% of the classes contain six bugs, 0.3109% of classes contain seven and eleven bugs, 0.5181% of classes contain eight bugs, and 0.1036% of the class contain nine, thirteen, seventeen, eighteen, and twenty-eight bugs.

6.2. Metrics Data. CK metric values for WMC, DIT, NOC, CBO, RFC, and LCOM, respectively, for AIF version 1.6 are graphically represented in Figures 5, 6, 7, 8, 9, and 10.

6.3. Descriptive Statistics and Correlation Analysis. This subsection gives the comparative analysis of the fault data, descriptive statistics of classes, and the correlation among the six metrics with that of Basili et al. [1]. Basili et al. studied object-oriented systems written in C++ language. They carried out an experiment in which they set up eight project groups each consisting of three students. Each group had the same task of developing small/medium-sized software system. Since all the necessary documentation (for instance, reports about faults and their fixes) were available, they could search for relationships between fault density and metrics. They used the same CK metric suite. Logistic regression was employed to analyze the relationship between metrics and the fault proneness of classes.

The obtained CK metric values of AIF version 1.6 are compared with the results of Basili et al. [1]. In comparison with Basili, the total number of classes considered is much greater; that is, 965 classes were considered (Vs. 180). Table 6 shows the comparative statistical analysis results obtained for

Table 6: Descriptive statistics of classes.

	WMC	DIT	NOC	CBO	RFC	LCOM
Basili et al. [1]						
Max.	99.00	9.00	105.00	13.00	30.00	426.00
Min.	1.00	0.00	0.00	0.00	0.00	0.00
Median	9.50	0.00	19.50	0.00	5.00	0.00
Mean	13.40	1.32	33.91	0.23	6.80	9.70
Std Dev.	14.90	1.99	33.37	1.54	7.56	63.77
AIF version 1.6						
Max.	166.00	6.00	39.00	448.00	322.00	13617
Min.	0.00	0.00	0.00	0.00	0.00	0.00
Median	5.00	1.00	0.00	7.00	14.00	4.00
Mean	8.57	1.95	0.052	11.10	21.42	79.33
Std Dev.	11.20	1.27	2.63	22.52	25.00	523.75

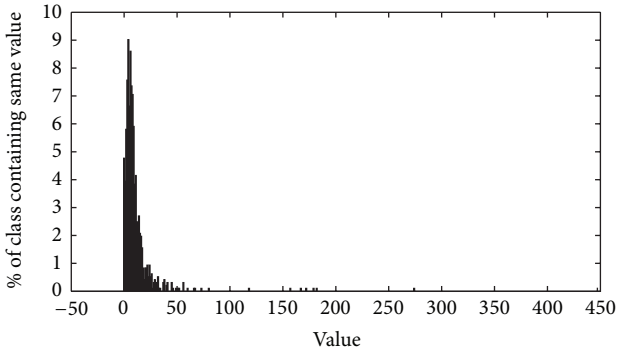

Figure 8: CBO of AIF version 1.6.

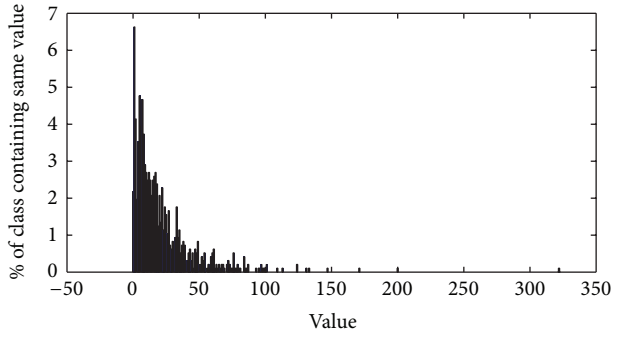

Figure 9: RFC of AIF version 1.6.

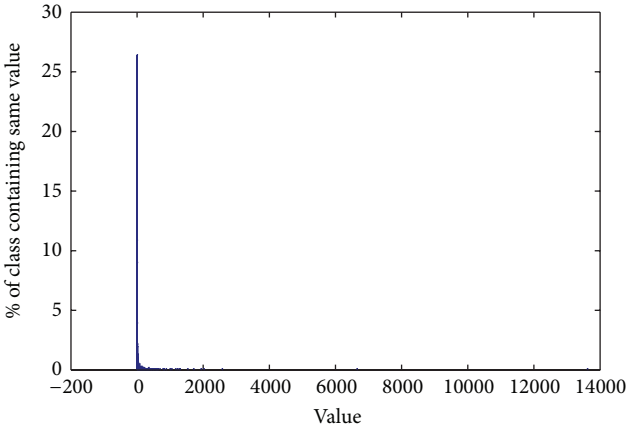

Figure 10: LCOM of AIF version 1.6.

Basili et al. and AIF version 1.6 for CK metrics indicating Max, Min, Median, and Standard deviation.

The dependency between CK metrics is computed using Pearson's correlations (R^2: coefficient of determination) and compared with Basili et al. [1] for AIF version 1.6. The coefficient of determination, R^2, is useful because it gives the proportion of the variance (fluctuation) of one variable that is predictable from the other variable. It is a measure that allows a researcher to determine how certain one can be in making predictions from a certain model/graph. Table 7 shows the Pearson's correlations for the data set used by Basili et al. [1] and the correlation metrics of AIF version 1.6.

From Table 7, w.r.t AIF version 1.6, it is observed that correlation between WMC and RFC is 0.77 which is highly correlated; that is, these two metrics are very much linearly dependent on each other. Similarly, correlation between WMC and DIT is 0, which indicates that they are loosely correlated; that is, there is no dependency between these two metrics.

6.4. Fault Prediction Using Statistical Methods

6.4.1. Linear Regression Analysis. Table 8 shows results obtained for linear regression analysis, in which the fault is considered as the dependent variable and the CK metrics are the independent variables.

"R" represents the coefficient of correlation; "P" refers to the significance of the metric value. If $P < 0.001$, then the metrics are of very great significance in fault prediction.

6.4.2. Logistic Regression Analysis. The logistic regression method helps to indicate whether a class is faulty or not but does not convey anything about the possible number of faults in the class. Univariate and multivariate logistic regression techniques are applied to predict whether the

Table 7: Correlations between metrics.

	WMC	DIT	NOC	CBO	RFC	LCOM
Basili et al. [1]						
WMC	1.00	0.02	0.24	0.00	0.13	0.38
DIT		1.00	0.00	0.00	0.00	0.01
NOC			1.00	0.00	0.00	0.00
CBO				1.00	0.31	0.01
RFC					1.00	0.09
LCOM						1.00
AIF version 1.6						
WMC	1.00	0.00	0.03	0.10	0.77	0.60
DIT		1.00	0.00	0.00	0.00	0.01
NOC			1.00	0.024	0.025	0.027
CBO				1.00	0.08	0.05
RFC					1.00	0.42
LCOM						1.00

Table 8: Linear regression analysis.

Version	R	P value	Std. error
1.2	0.5360	0.000	0.1114
1.4	0.5024	0.000	0.1450
1.6	0.5154	0.000	0.0834

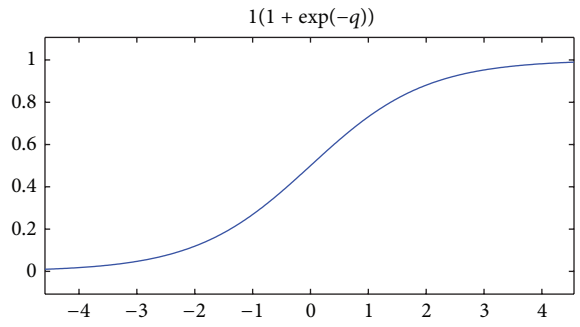

Figure 11: Logistic graph.

class is faulty or not. Univariate regression analysis is used to examine the effect of each metric on fault of the class while multivariate regression analysis is used to examine the common effectiveness of metrics on fault of the class. The results of three versions of AIF are compared considering these two statistical techniques. Figure 11 shows the typical "S" curve obtained (similar to Sigmoid function) for the AIF version 1.6 using multivariate logistic regression. Tables 9 and 10 contain the tabulated values for the results obtained by applying univariate and multivariate regression analysis, respectively.

From Table 9, it can be observed that all metrics of CK suite are highly significant except for DIT. The P value for the three versions (w.r.t DIT) is 0.335, 0.108, and 0.3527, respectively. Higher values of "P" are an indication of less significance.

Univariate and multivariate logistic regression statistical methods were used for classifying a class as faulty or not faulty. Logistic regression was applied with a threshold value 0.5; that is, $\pi > 0.5$ indicates that a class is classified as "faulty," otherwise it is categorized as "not faulty" class.

Tables 11 and 12 represent the confusion matrix for number of classes with faults before and after applying regression analysis, respectively, for AIF version 1.6. From Table 11 it is clear that before applying the logistic regression, a total number of 777 classes contained zero bugs and 188 classes contained at least one bug. After applying logistic regression (Table 12), a total of 767 + 16 classes are classified correctly with accuracy of 81.13%.

The performance parameters of all three versions of the AIF are shown in Table 13, obtained by applying univariate and multivariate logistic regression analysis. Here precision, correctness, completeness, and accuracy [1, 13, 27, 28] are taken as a performance parameters. By using multivariate logistic regression, accuracy of AIF version 1.2 is found to be 64.44%, accuracy of AIF version 1.4 is 83.37%, and that of AIF version 1.6 is 81.13%.

From the results obtained by applying linear and logistic regression analysis, it is found that out of the six metrics WMC appears to have more impact in predicting faults.

6.5. Fault Prediction Using Neural Network

6.5.1. Artificial Neural Network. ANN is an interconnected group of nodes. In this paper, three layers of ANN are considered, in which six nodes act as input nodes, nine nodes represent the hidden nodes, and one node acts as output node.

ANN is a three-phase network; the phases are used for learning, validation and testing purposes. So in this article, 70% of total input pattern is considered for learning phase, 15% for validation, and the rest 15% for testing. The regression analysis carried out classifies whether a class is faulty or not faulty. The prediction models of ANN and its forms such as

Table 9: Analysis of univariate regression.

	Coefficient			Constant			*P* value			*R* value		
	ver 1.2	ver 1.4	ver 1.6	ver 1.2	ver 1.4	ver 1.6	ver 1.2	ver 1.4	ver 1.6	ver 1.2	ver 1.4	ver 1.6
WMC	0.028	0.05	0.03	−0.83	−2.11	−1.77	0.0013	0.0007	0.00	0.130	0.240	0.18
DIT	−0.067	0.10	0.05	−0.46	−1.83	−1.53	0.335	0.108	0.3257	−0.039	0.054	0.02
NOC	0.137	0.09	0.13	−0.66	−1.67	−1.50	0.0007	0.00	0.00	0.136	0.13	0.16
CBO	0.011	0.01	0.02	−0.71	−1.80	−1.66	0.017	0.00	0.00	0.096	0.15	0.17
RFC	0.012	0.02	0.01	−0.86	−2.15	−1.79	0.0014	0.00	0.00	0.130	0.23	0.17
LCOM	0.007	0.007	0.007	−0.64	−1.67	−1.48	0.0349	0.0004	0.0007	0.085	0.11	0.11

Table 10: Multivariate logistic regression analysis.

	Coefficient		
	AIF version 1.2	AIF version 1.4	AIF version 1.6
WMC	0.0195	0.0574	0.0320
DIT	−0.041	0.000	0.000
NOC	0.1231	0.000	0.000
CBO	0.005	0.008	0.001
RFC	0.0071	0.0081	0.0109
LCOM	0	−0.001	0
Constant	−0.917	−2.785	−2.157

Table 11: Before applying regression.

	Not-faulty	Faulty
Not-Faulty	777	0
Faulty	188	0

Table 12: After applying regression.

	Not-faulty	Faulty
Not-Faulty	767	10
Faulty	172	16

PNN, RBFN, and FLANN, not only classify the class as faulty or not faulty but also highlight the number of bugs found in the class and these bugs are fixed in the testing phase of software development life cycle.

In this paper six CK metrics are taken as input, and output is the fault prediction accuracy rate required for developing the software. The network is trained using Gradient descent method and Levenberg Marquardt method.

(a) Gradient Descent Method. Gradient descent method is used for updating the weights using (15) and (16). Table 14 shows the performance metrics of AIF version 1.6. Figure 12 shows the graph plot for variation of mean square error values w.r.t no of epoch (or iteration) for AIF version 1.6.

(b) Levenberg Marquardt Method. Levenberg Marquardt method [21, 22] is a technique for updating weights. In case of Gradient descent method, learning rate α is constant but in Levenberg Marquardt method, learning rate α varies in every iteration. So this method consumes less number of iterations to train the network. Table 15 shows the performance metrics for AIF version 1.6 using Levenberg Marquardt method.

Figure 13 shows the graph plot for variation of mean square error values w.r.t number of epoch for AIF version 1.6.

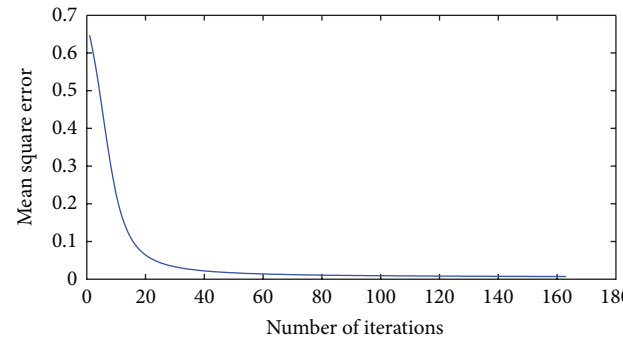

Figure 12: MSE versus number of epoch w.r.t Gradient descent NN.

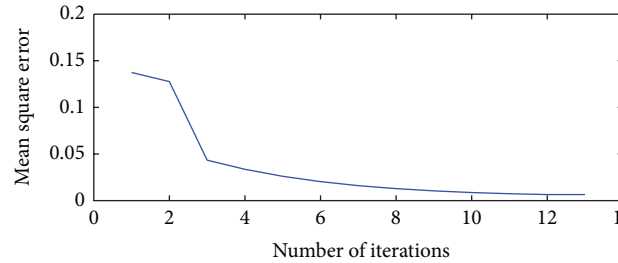

Figure 13: MSE versus number of epoch w.r.t Levenberg-marquardt NN.

Table 13: Precision, correctness, completeness, and accuracy for three versions of AIF.

	Precision (%)			Correctness (%)			Completeness (%)			Accuracy (%)		
	ver 1.2	ver 1.4	ver 1.6	ver 1.2	ver 1.4	ver 1.6	ver 1.2	ver 1.4	ver 1.6	ver 1.2	ver 1.4	ver 1.6
WMC	61.11	41.17	57.14	61.11	41.17	57.14	5.09	4.82	4.25	66.13	84.02	**81.71**
DIT	—	—	—	—	—	—	0	0	0	64.47	83.37	80.51
NOC	75	75	66.66	75	75	66.66	5.55	2.06	5.31	65.78	83.6	81.03
CBO	60	57.14	77.77	60	57.14	77.77	2.77	2.75	3.72	64.8	83.48	81.03
RFC	66.66	36.36	50	66.66	36.36	50	4.62	2.75	2.12	65.29	83.02	80.51
LCOM	66.66	50	60	0.66	0.5	0.6	2.77	6.8	1.59	64.96	83.37	80.62
MULTI	68.75	50	61.53	68.75	50	61.53	10.18	7.58	8.51	66.44	83.37	81.13

Table 14: Accuracy prediction using gradient descent NN.

MAE	MARE	RMSE	R	P value	Std. error	Accuracy (%)
0.0594	1.093	0.0617	−0.2038	0.0044	0.0048	94.0437

Table 15: Accuracy prediction using Levenberg Marquardt.

MAE	MARE	RMSE	R	P value	Std. error	Accuracy (%)
0.0023	1.1203	0.0308	−0.2189	0.0022	0.0041	90.4977

Table 16: Accuracy prediction using FLANN.

MAE	MARE	RMSE	R	P value	Std. error	Accuracy (%)
0.0304	0.7097	0.0390	0.3308	$2.4601e-06$	0.0050	96.3769

Table 17: Accuracy prediction using basic RBFN.

MAE	MARE	RMSE	R	P value	Std. error	Accuracy (%)
0.0279	0.3875	0.0573	0.1969	0.059	0.006	97.2792

6.5.2. Functional Link Artificial Neural Network (FLANN). FLANN architecture for software fault prediction is a single layer feed-forward neural network consisting of an input and output layer. FLANN does not incorporate any hidden layer and hence has less computational cost. In this paper, adaptive algorithm has been used for updating the weights as shown in (21). Figure 14 shows the variation of mean square values against number of epochs for AIF version 1.6. Table 16 shows the performance metrics of FLANN.

6.5.3. Radial Basis Function Network. In this paper, Gaussian radial function is used as a radial function. Gradient descent learning and hybrid learning methods are used for updating the centers and weights, respectively.

Three layered RBFN has been considered, in which six CK metrics are taken as input nodes, nine hidden centers are taken as hidden nodes, and output is the fault prediction rate. Table 17 shows the performance metrics for AIF version 1.6.

(a) Gradient Descent Learning Method. Equations (25) and (26) are used for updating center and weight during training phase. After simplifying (25), the equation is represented as:

$$C_{ij}(k+1) = C_{ij}(k) - \eta_1 \left(y' - y\right) Wi \frac{\phi_i}{\sigma^2} \left(x_j - C_{ij}(k)\right) \quad (39)$$

and the modified Equation (26) is formulated as:

$$W_i(k+1) = W_i(k) + \eta_2 \left(y' - y\right) \phi_i, \quad (40)$$

where σ is the width of the center and k is the current iteration number. Table 18 shows the performance metrics for AIF version 1.6. Figure 15 indicates the variation of MSE w.r.t number of epochs.

(b) Hybrid Learning Method. In Hybrid learning method, centers are updated using (27) while weights are updated using supervised learning methods. In this paper, least mean square error (LMSE) algorithm is used for updating the weights. Table 19 shows the performance matrix for AIF version 1.6. Figure 16 shows the graph for variation of MSE versus number of epochs.

6.5.4. Probabilistic Neural Network (PNN). As mentioned in Section 4.2.4, PNN is a multilayered feed-forward network with four layers such as input, hidden, summation, and output layer.

In PNN, 50% of faulty and nonfaulty classes are taken as input for hidden layers. Gaussian elimination (28) is used as a hidden node function. The summation layers sum

Table 18: Accuracy prediction using RBFN gradient.

MAE	MARE	RMSE	R	P value	Std. Error	Accuracy (%)
0.0207	0.2316	0.0323	0.3041	$1.6302e-05$	0.0041	97.2475

Table 19: Accuracy prediction using hybrid RBFN.

MAE	MARE	RMSE	R	P value	Std. Error	Accuracy (%)
0.0614	0.1032	0.0316	0.9184	$3.1834e-79$	0.0013	98.4783

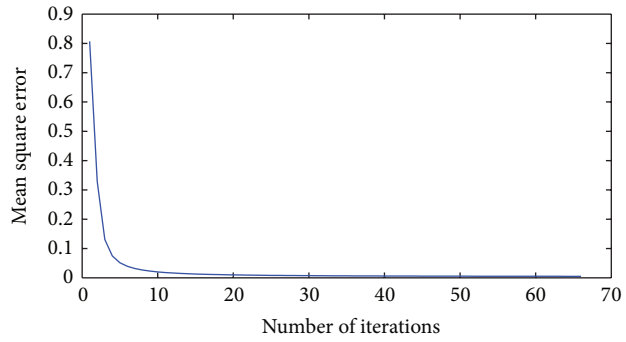

Figure 14: Graph plot for MSE versus number of iterations (epoch) w.r.t FLANN.

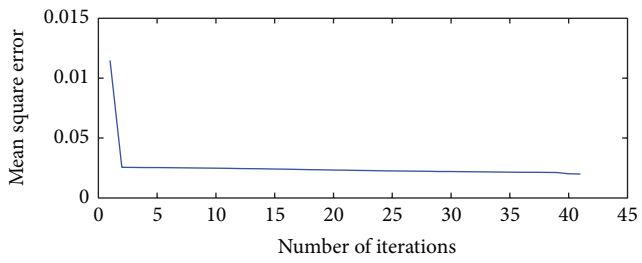

Figure 15: MSE versus number of epochs w.r.t gradient RBFN.

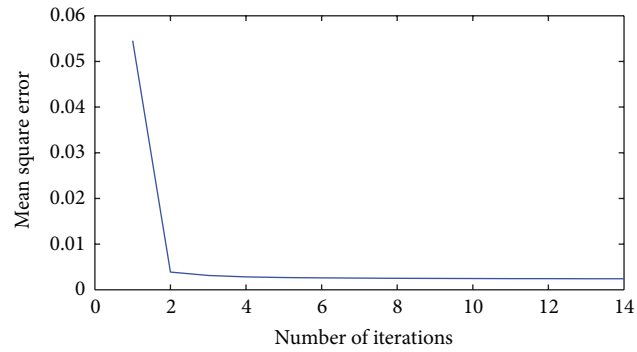

Figure 16: MSE versus number of epochs w.r.t hybrid RBFN.

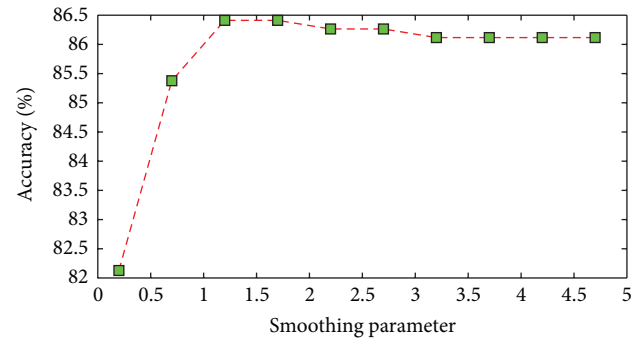

Figure 17: Accuracy rate versus smoothing parameter.

contribution of each class of input patterns and produce a net output which is a vector of probabilities. The output pattern having maximum summation value is classified into respective class. Figure 17 shows the variation of accuracy for different values of smoothing parameter.

6.6. Comparison. Table 20 shows the tabulated results for the obtained performance parameter values, number of epochs, and accuracy rate by applying three neural network techniques. This performance table is an indication of better fault prediction model. In this comparative analysis, the performance parameter mean square error (MSE) was taken as a criterion to compute the performance parameters (such as MARE, MSE, number of epochs, and accuracy rate) when four neural network techniques were applied. During this process the MSE value of 0.002 was set a threshold for evaluation. Based on the number of iterations and the accuracy rate obtained by the respective NN technique, best prediction model was determined.

From Table 20 it is evident that gradient NN method obtained an accuracy rate of 94.04% in 162 epochs (iterations). LM technique, which is an improvised model of ANN, obtained 90.4% accuracy rate. This accuracy rate is less than gradient NN but this approach (LM method) took only 13 epochs. PNN method achieved a classification rate of 86.41%.

The three types of RBFN, namely, basic RBFN, gradient, and hybrid methods obtained a prediction rate of 97.27%, 97.24%, and 98.47%, respectively. Considering the number of epochs, RBFN hybrid method obtained better prediction rate of 98.47% in only 14 epochs when compared with gradient method (41 epochs) and basic RBFN approaches.

FLANN architecture obtained 96.37% accuracy rate with less computational cost involved. FLANN obtained accuracy rate in 66 epochs as it has no hidden layer involved in its architecture.

Table 20: Performance metrics.

	Performance parameters					
AI technique	Epoch	MAE	MARE	RMSE	Std. Error	Accuracy
Gradient descent	162	0.0594	1.0930	0.0617	0.0048	94.04
LM	13	0.0023	1.1203	0.0308	0.0041	90.49
RBFN basic	—	0.0279	0.3875	0.0573	0.06	97.27
RBFN gradient	41	0.0207	0.2316	0.0323	0.0041	97.24
RBFN hybrid	**14**	0.0614	0.1032	0.0316	0.0013	**98.47**
FLANN	66	0.0304	0.7097	0.0390	0.0050	96.37

The performance of PNN is shown in Figure 17. Highest accuracy in prediction was obtained for smoothing parameter value of 1.7. PNN obtained a classification rate of 86.41%.

RBFN using hybrid learning model gives the least values for MAE, MARE, RMSE, and high accuracy rate. Hence, from the obtained results by using ANN techniques it can be concluded that RBFN hybrid approach obtained the best fault prediction rate in less number of epochs when compared with other three ANN techniques.

7. Conclusion

System analyst use of prediction models to classify fault prone classes as faulty or not faulty is the need of the day for researchers as well as practitioners. So, more reliable approaches for prediction need to be modeled. In this paper, two approaches, namely, statistical methods and machine learning techniques were applied for fault prediction. The application of statistical and machine learning methods in fault prediction requires enormous amount of data and analyzing this huge amount of data is necessary with the help of a better prediction model.

This paper proposes a comparative study of different prediction models for fault prediction for an open-source project. Fault prediction using statistical and machine learning methods were carried out for AIF by coding in MATLAB environment. Statistical methods such as linear regression and logistic regression were applied. Also machine learning techniques such as artificial neural network (gradient descent and Levenberg Marquardt methods), Functional link artificial neural network, radial basis function network (RBFN basic, RBFN gradient, and RBFN hybrid), and probabilistic neural network techniques were applied for fault prediction analysis.

It can be concluded from the statistical regression analysis that out of six CK metrics, WMC appears to be more useful in predicting faults. Table 20 shows that hybrid approach of RBFN obtained better fault prediction in less number of epochs (14 iterations) when compared with the other three neural network techniques.

In future, work should be replicated to other open-source projects like Mozilla using different AI techniques to analyze which model performs better in achieving higher accuracy for fault prediction. Also, fault prediction accuracy should be measured by combining multiple computational intelligence techniques.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751–761, 1996.

[2] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, 1976.

[3] M. H. Halstead, *Elements of Software Science*, Elsevier Science, New York, NY, USA, 1977.

[4] W. Li and S. Henry, "Maintenance metrics for the Object-Oriented paradigm," in *Proceedings of the 1st International Software Metrics Symposium*, pp. 52–60, 1993.

[5] S. R. Chidamber and C. F. Kemerer, "Metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[6] F. B. E. Abreu and R. Carapuca, "Object-Oriented software engineering: measuring and controlling the development process," in *Proceedings of the 4th International Conference on Software Quality*, pp. 1–8, McLean, Va, USA, October 1994.

[7] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics*, Prentice Hall, Englewood, NJ, USA, 1994.

[8] R. Martin, "OO design quality metrics—an analysis of dependencies," in *Proceedings of the Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics (OOPSLA '94)*, 1994.

[9] D. P. Tegarden, S. D. Sheetz, and D. E. Monarchi, "A software complexity model of object-oriented systems," *Decision Support Systems*, vol. 13, no. 3-4, pp. 241–262, 1995.

[10] W. Melo and F. B. E. Abreu, "Evaluating the impact of object-oriented design on software quality," in *Proceedings of the 3rd International Software Metrics Symposium*, pp. 90–99, Berlin, Germany, March 1996.

[11] L. Briand, P. Devanbu, and W. Melo, "Investigation into coupling measures for C++," in *Proceedings of the IEEE 19th International Conference on Software Engineering Association for Computing Machinery*, pp. 412–421, May 1997.

[12] L. Etzkorn, J. Bansiya, and C. Davis, "Design and code complexity metrics for OO classes," *Journal of Object-Oriented Programming*, vol. 12, no. 1, pp. 35–40, 1999.

[13] L. C. Briand, J. Wüst, J. W. Daly, and D. Victor Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," *The Journal of Systems and Software*, vol. 51, no. 3, pp. 245–273, 2000.

[14] M.-H. Tang, M.-H. Kao, and M.-H. Chen, "Empirical study on object-oriented metrics," in *Proceedings of the 6th International Software Metrics Symposium*, pp. 242–249, November 1999.

[15] K. El Emam, W. Melo, and J. C. Machado, "The prediction of faulty classes using object-oriented design metrics," *Journal of Systems and Software*, vol. 56, no. 1, pp. 63–75, 2001.

[16] T. M. Khoshgoftaar, E. B. Allen, J. P. Hudepohl, and S. J. Aud, "Application of neural networks to software quality modeling of a very large telecommunications system," *IEEE Transactions on Neural Networks*, vol. 8, no. 4, pp. 902–909, 1997.

[17] R. Hochman, T. M. Khoshgoftaar, E. B. Allen, and J. P. Hudepohl, "Evolutionary neural networks: a robust approach to software reliability problems," in *Proceedings of the 8th International Symposium on Software Reliability Engineering (ISSRE '97)*, pp. 13–26, November 1997.

[18] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, "The PROMISE Repository of empirical software engineering data," West Virginia University, Department of Computer Science, 2012, http://promisedata.googlecode.com.

[19] Y. Kumar Jain and S. K. Bhandare, "Min max normalization based data perturbation method for privacy protection," *International Journal of Computer and Communication Technology*, vol. 2, no. 8, pp. 45–50, 2011.

[20] R. Battiti, "First and Second-Order Methods for Learning between steepest descent and newton's method," *Neural Computation*, vol. 4, no. 2, pp. 141–166, 1992.

[21] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, no. 2, pp. 164–168, 1944.

[22] D. W. Marquardt, "An algorithm for the lest-squares estimation of non-linear parameters," *SIAM Journal of Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[23] Y. H. Pao, *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Reading, UK, 1989.

[24] D. F. Specht, "Probabilistic neural networks," *Neural Networks*, vol. 3, no. 1, pp. 109–118, 1990.

[25] C. Catal, "Performance evaluation metrics for software fault prediction studies," *Acta Polytechnica Hungarica*, vol. 9, no. 4, pp. 193–206, 2012.

[26] X. Yaun, T. M. Khoshgoftaar, E. B. Allen, and K. Ganesan, "Application of fuzzy clustering to software quality prediction," in *Proceedings of the 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSEST '00)*, pp. 85–91, March 2000.

[27] T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897–910, 2005.

[28] G. Denaro, M. Pezzè, and S. Morasca, "Towards industrially relevant fault-proneness models," *International Journal of Software Engineering and Knowledge Engineering*, vol. 13, no. 4, pp. 395–417, 2003.

[29] S. Kanmani and U. V. Rymend, "Object-Oriented software quality prediction using general regression neural networks," *SIGSOFT Software Engineering Notes*, vol. 29, no. 5, pp. 1–6, 2004.

[30] N. Nagappan and W. Laurie, "Early estimation of software quality using in-process testing metrics: a controlled case study," in *Proceedings of the 3rd Workshop on Software Quality*, pp. 1–7, St. Louis, Mo, USA, 2005.

[31] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly Iterative or agile software development processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 402–419, 2007.

[32] K. K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study," *Software Process Improvement and Practice*, vol. 14, no. 1, pp. 39–62, 2009.

[33] F. Wu, "Empirical validation of object-oriented metrics on NASA for fault prediction," in *Proceedings of theInternational Conference on Advances in Information Technology and Education*, pp. 168–175, 2011.

[34] H. Kapila and S. Singh, "Analysis of CK metrics to predict software fault-proneness using bayesian inference," *International Journal of Computer Applications*, vol. 74, no. 2, pp. 1–4, 2013.