Software Fault Prediction using Object-Oriented Metrics

Lov Kumar

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769008, Odisha, India
June 2014

Software Fault Prediction using Object-Oriented Metrics

Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Technology

in

Computer Science and Engineering

(Specialization: Software Engineering)

by

Lov Kumar

(Roll No.- 212CS3371)

under the supervision of

Prof. S. K. Rath

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela, Odisha, 769 008, India

June 2014

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

Certificate

This is to certify that the work in the thesis entitled ***Software Fault Prediction using Object-Oriented Metrics*** by ***Lov Kumar*** is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Master of Technology with the specialization of Software Engineering in the department of Computer Science and Engineering, National Institute of Technology Rourkela. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Place: NIT Rourkela
Date: June 1, 2014

(Prof. Santanu Ku. Rath)
Professor, CSE Department
NIT Rourkela, Odisha

Acknowledgment

I am grateful to numerous local and global peers who have contributed towards shaping this thesis. At the outset, I would like to express my sincere thanks to Prof. Santanu Ku. Rath for his advice during my thesis work. As my supervisor, he has constantly encouraged me to remain focused on achieving my goal. His observations and comments helped me to establish the overall direction to the research and to move forward with investigation in depth. He has helped me greatly and been a source of knowledge.

I am very much indebted to Prof. Santanu Ku. Rath, for his continuous encouragement and support. He is always ready to help with a smile. I am also thankful to all the professors at the department for their support.

I would like to thank all my friends and lab-mates for their encouragement and understanding. Their help can never be penned with words.

I must acknowledge the academic resources that I have got from NIT Rourkela. I would like to thank administrative and technical staff members of the Department who have been kind enough to advise and help in their respective roles.

Last, but not the least, I would like to dedicate this thesis to my family, for their love, patience, and understanding.

Lov kumar

Roll-212cs3371

Abstract

Fault-prediction techniques aim to predict the fault prone software modules in order to streamline the effort to be applied in the later phases of software development. Many fault-prediction techniques have been proposed and evaluated for their performance using various performance criteria. However, due to the lack of compiling their performances in proper perspective, one significant issue about the viability of these techniques has not been adequately addressed. In this study, an adaptive cost evaluation framework is proposed that incorporates cost drivers for various fault removal phases, and performs a cost-benefit analysis for the misclassification of faults. Accordingly, our study focuses on investigating two important and related research questions regarding the viability of fault prediction. First, for a given software product, whether performing fault prediction analysis is economically effective or not?. In case of an positive affirmation, then emphasis is provided on how to choose a fault prediction technique for an overall improved performance in terms of cost-effectiveness. In this study, Object-Oriented software metrics have been considered to provide requisite input data to design a classifier using statistical, machine learning and hybrid methods of soft computing. This work, also extends the study on finding the effectiveness of feature reduction techniques. From the obtained results, it is observed that performing fault prediction is quite desirable for those software systems, when the percentage of faulty modules are below the range of certain threshold value.

Keywords: ANN, ANGA, CSA, GA, linear regression, logistics regression, MNPSO, NGA, NCSA, NPSO, Naive Bayes, polynomial regression, PCA, PSO, SVM, RSA, Software fault estimation, software metrics.

Contents

List of Figures

List of Tables

List Of Abbreviation

ANN	Artificial neural network
DIT	Depth of inheritance tree
CBO	Coupling between objects
CSA	Clonal Selection Algorithm
E_{cost}	Estimate fault removal cost with using fault prediction
FP	False Positive
FN	False Negative
GA	Genetics algorithm
LCOM	Lack of cohesion among methods
NE_{cost}	Normalized Estimated fault removal cost
NOA	Number of attributes
NOAI	Number of attributes inherited
NOC	Number of children
NLOC	Number of line of codes
NOM	Number of methods
NOMI	Number of methods inherited
NOPA	Number of private attribute
N$\bar{P}$OA	Number of public attribute
NOPM	Number of private methods
N$\bar{P}$OM	Number of public methods
PCA	Principal components analysis
PSO	Particle Swarm Optimization
RSA	Rough set analysis
RFC	Response for class
SVM	Support Vector Machine
TP	True Positive
TN	True Negative
T_{cost}	Estimate fault removal cost with using testing
WMC	Weighted method per class

CHAPTER 1

Introduction

Literature Review

Software metrics

Performance Parameter

Motivation

Thesis Organization

Chapter 1

Introduction

Fault prediction is necessary in software development life cycle in order to reduce the probable software failure and is carried out mostly during initial planning to identify fault-prone modules. Fault prediction not only gives an insight to the need for increased quality of monitoring during software development but also provides necessary tips to undertake suitable verification and validation approaches that eventually lead to improvement of efficiency and effectiveness of fault prediction. Effectiveness of a fault prediction is studied by applying a part of previously known data related to faults and predicting its performance against other part of the fault data. Several researchers have worked on building prediction models for software fault prediction but less emphasis has been given on the study of effectiveness of fault prediction.

Present day software development is mostly desired to be based on Object-Oriented (OO) paradigm. The quality of OO 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. Some of the software metrics available in literature are as follows: Abreu MOOD metric suite [1], Bansiya and Davis (QMOOD metrics suite) [2], Bieman and Kang [3], Briand *et al.* [4], Etzkorn *et al.* [5], Halstead [6], Henderson-sellers [7], Li and Henry [8], McCabe [9], Tegarden *et al.* [10], Lorenz and Kidd [11] and CK metric [12] suite.

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 quality of the developed software. In practice, software quality mainly refers

to FURPS model such as functionality, usability, reliability, Portability and supportability. This study mostly focus on the aspect of improving reliability of a software by reducing the number of faults in the software.

In order to estimate the reliability of a class, several traditional methods are available in literature. But less importance has been given on using machine learning techniques. Artificial intelligence techniques, a subset of machine learning methods have the ability of computer, software and firmware to measure the properties of a class, that human beings recognize as intelligent behavior. These methods are able to approximate the non-linear function with more precision. Hence they can be applied for quality estimation in order to achieve better accuracy.

1.1 Literature Review

This section presents a review of literature on the application of software metrics. Table 1.1 shows the summary of Empirical Literature on software metrics.

Basili *et al.,* [13] experimentally analyzed the impact of CK metric suite in fault prediction. Briand *et al.,* [14] found out the relationship between fault and the metrics using univariate and multivariate logistic regression models. Tang *et al.,* [16] investigated the dependency between CK metric suite and the Object-Oriented system faults. Emam *et al.,* [18] conducted empirical validation on Java application and found that export coupling has great influence with faults. Khoshgoftaar *et al.,* [21], Hochman [22] conducted experimental analysis on telecommunication model and found that artificial neural network (ANN) model is give accurately output than other discriminant model. In their approaches, nine software metrics were used for modules developed in procedural paradigm. Since then, ANN models have taken a rise in their usage for prediction modeling. Hence, in this study, different ANN models are used for fault prediction of embedded software.

Also few researchers have presented cost based evaluation models for predicting the effectiveness of fault prediction. In this section, the study related to the measure of cost effectiveness for fault prediction has been tabulated in Table 1.2.

Table 1.1: Summary of literature on reliability prediction using software Metrics suite

Study	Software Metrics tested	Dependent variable	Summary of Results
Basili *et al.* (1996) [13]	All CK metrics	Fault Proneness	Correlated WMC, DIT, NOC, RFC and CBO with defects for eight academic projects.
Chidamber *et al.* (1998) [14]	All CK metrics	design effort, rework effort and productivity	Found that Low LCOM and High CBO accounted for lower productivity, greater rework and design effort in case of three financial service applications.
Briand *et al.* (1999) [15]	CBO, RFC, LCOM	Fault Proneness	Found that three CK metrics i.e., CBO, RFC and LCOM were found to be associated with fault proneness of classes for an industrial case study.
Tang *et al.* (1999) [16]	WMC, RFC	Fault Proneness	Found higher WMC and RFC were found to be associated with fault proneness. They are utilized real time systems for testing and maintenance.
Briand *et al.* (2000) [4]	All CK metrics	Fault Proneness	Observed that classes with higher WMC, CBO, DIT and RFC were more fault prone while classes having more children (NOC) were less fault prone. LCOM did not account for the defects associated with the eight academic projects studied in this analysis.
Cartwright and Shepperd (2000) [17]	DIT, NOC	Defect density	Observed that DIT and NOC influence defect density in case of medium sized telecommunication system.
El Emam *et al.* (2001b) [18]	All CK metrics	Fault Proneness	Found that size confound the effect of all metrics on fault proneness for large telecommunication application.
Ramanath Subramanyam and M.S. Krishnan (2003) [19]	WMC, CBO, DIT	Defects	They find that the effects of these metrics on defects vary across the samples from two programming languages C++ and Java.
Olague *et al.* (2007) [20]	All CK metrics	Fault Proneness	They explore the ability of three metrics i.e., CK metrics, MOOD and QMOOD suites to predict fault-prone classes using defect data for six versions of Rhino, an open-source implementation of JavaScript written in Java.

Table 1.2: Fault prediction effectiveness based on Cost evaluation model

Author	**Cost evaluation criteria**
Jiang *et al.,* [23]	Introduced cost curve based on Receiver Operating Characteristic (ROC).
Mende *et al.,* [24]	Introduced a performance measure (P_{opt}) and compared prediction model with an optimal model. P_{opt} accounted module size to evaluate the performance of a fault-prediction technique.
Mende *et al.,* [25]	Proposed two strategies namely AD (effort-aware binary prediction) and DD (effort-aware prediction based on defect density) to include the notion of effort awareness into fault-prediction techniques.
Arisholm *et al.,* [26]	Proposed a cost performance measure - Cost Effectiveness (CE), a variation of lift charts where the x-axis contains the ratio of lines of code instead of modules.

In this study, linear regression, polynomial regression, logistic regression, Naive Bayes and SVM models have been considered so as to predict software quality by classifying a class as faulty or not faulty .

In literature, classification models are mostly built using statistical analysis. Neural networks (NN) have seen an explosion of interest over the years, and are being successfully applied across a range of problem domains. When the problems of classification, prediction, NN are being used, NN can be used as a technique to design prediction model because it is a very sophisticated modeling technique that enables modeling of complex function. In this thesis work, software metrics has been considered for quality estimation using various statistical and artificial intelligence techniques.

1.2 Software Metrics

A software metric is the measurement of a individual characteristic of a program's efficiency or performance and also used to measures the attributes of software products and processes. At present, software development based on Object-Oriented (OO) Paradigm is becoming more and more pronounced. The Object-Oriented

paradigm for the software development differs from traditional procedural so the traditional metrics can not be applied on OO software.

A number of OO software metrics have been proposed by researchers and practitioners to evaluate the quality of OO software. The most commonly used metric suites are: Abreu MOOD metric suite [1], Bansiya and Davis (QMOOD metrics suite) [2], Bieman and Kang [3], Briand *et al.* [4], Etzkorn *et al.* [5], Halstead [6], Henderson-sellers [7], Li and Henry [8], McCabe [9], Tegarden *et al.* [10], Lorenz and Kidd [11] and CK metric [12] suite. Table 1.3 gives the basic definitions of software metric.

Table 1.3: Software metrics

Software Metric	**Description**
WMC	Summation of the complexities of all class methods
NOC	Number of immediate sub-classes subordinate to a class in the class hierarchy
DIT	Maximum height of the class hierarchy
CBO	Number of other classes to which it is coupled
RFC	A set of methods that can potentially be executed in response to a message received by an object of that class
LCOM	Measures the dissimilarity of methods in a class via instanced variables
NOM	Number of methods defined in a class
NOA	Number of attribute defined in a class
NOAI	Counts the number of attribute which are inherited by all member subclasses.
NOMI	Counts the number of method which are inherited by all member subclasses.
Fan-in	It defines as the summation of number of local flows into that procedure and the number of data structures from which that procedure retrieves information.
Fan-out	It defines as the summation of number of local flows out of that procedure and the number of data structures that the procedure updates
NOPM	Number of private methods in a class
NOPA	Number of private attribute in a class
NOPM	Number of public methods in a class
NOPA	Number of public attribute in a class
NLOC	it is used to measure the size of a program by counting the number of lines in the text of the source code.

1.3 Performance evaluation parameters

The following sub-sections give the basic definitions of the performance parameters used for fault prediction.

Table 1.4: Confusion matrix to classify a class as faulty and not-faulty

	Non-Faulty	Faulty
Non-Faulty	True Negative (TN)	False Positive (FP)
Faulty	False Negative (FN)	True Positive (TP)

The confusion matrix are categories into four category :

i. True positives (TP) are the number of modules correctly classified as faulty modules.

ii. False positives (FP) refer to not-faulty classes incorrectly labeled as faulty classes.

iii. True negatives (TN) correspond to not-faulty modules correctly classified as such.

iv. Finally, false negatives (FN) refer to faulty classes incorrectly classified as not-faulty classes.

These are the performance parameter used to measures the classification techniques.

- Precision

  It is used to measure the degree to which the repeated measurements under unchanged conditions show the same results.

$$Precision = \frac{TP}{FP + TP} \tag{1.1}$$

- Recall

  Recall indicates the how many of the relevant item that are to be identified. it is represented as:

$$Recall = \frac{TP}{FN + TP} \tag{1.2}$$

- F-Measure

  F-Measure combine the precision and recall numeric value to give a single score, which is defined as the harmonic mean of the recall and precision. F-Measure is expressed as:

$$F - Measure = \frac{2 * Recall * Precision}{Recall + Precision} \tag{1.3}$$

- Specificity

  Specificity focus on how effectively a classifier identifies the negative labels. It is defined as:

$$Specificity = \frac{TN}{FP + TN} \tag{1.4}$$

- Accuracy

  Accuracy measure is the proportion of predicted fault prone modules that are inspected out of all modules. It is defined as:

$$Accuracy = \frac{TN + TP}{TP + TN + FP + FN} \tag{1.5}$$

1.4 Motivation

The majority of software bugs are small in nature, which cause large inconvenience that can be worked around by the user. Some noticeable cases wherein a simple mistake can affect millions and even cause injury and leads to loss of life. Software code, written by humans has a probability that every piece of software has fault or undocumented features. That is, the software does not meet the requirements. These faults can be due to bad design, problem misunderstanding, or just simple error just like a typo in a book. Unlike a book is read by a human who can usually infer the meaning of a misspelled word, the software is read by computers, which are comparatively stupid, and will perform what they are instructed to do. There are some major computer system failures caused by software bugs, such as:

- Nearly five patient deaths in the 1980's due to bugs in "Therac-25 radiation therapy" machine.
- In 1994, nine passengers are died in helicopter crashed in Chinook (Scotland) due to systems error in helicopter.
- In mar 2002, Britain's National Tax system overcharges 100,000 erroneous due to fault in their software system.
- In Japan's largest banks going off line for 24 hr, Internet banking services (IBS) being shut down for three days, delays in salary payments worth $1.5 billion into the accounts of 620,000 people and a backlog of more than 1 million unprocessed payments worth around $9 billion.
- In 2011, twenty two people wrongly arrested in Australia due to fault in new zealand $54.5 million courts computer system.
- In Apr 1992, first F-22 Raptor was crashed while landing at Edwards Air Force Base due to fault in flight controlling software system.
- In 2004, A2LL software which handling social services and unemployment in Germany transfer a payments to invalid account number due to failure in their system.

Thus, reducing these type of failure, Software fault prediction is one of the different strategies, which are conducted during the very beginning of software development life cycle. Fault prediction information not only for the increasing quality of software during the development but also give an information to understand suitable validation and verification activities in order to improve the effectiveness.

1.5 Thesis Organization

The rest of thesis is organized as follow:

- In Chapter-2, cost evaluation framework has been proposed which performs cost based analysis for misclassification of faults. This Chapter also focuses on investigating two important and related research questions regarding the viability of fault prediction. First, for a given software product, whether performing fault prediction analysis is economically effective or not?. In case of an positive affirmation, then emphasis is provided on how to choose a fault prediction technique for an overall improved performance in terms of cost-effectiveness

- In Chapter-3, artificial neural network (ANN), has been used to design a classifier, to classify a class as faulty and not faulty. In this chapter a case study of Eclipse JDT core has been considered for predicting the fault proneness.

- In Chapter-4, hybrid approach of artificial neural network and optimization algorithms i.e., genetics algorithm (GA), clonal selection algorithm (CSA) and Particle Swarm Optimization (PSO) have been used to design a classifier to classifying a class as faulty and not faulty. Here also same case study of Eclipse JDT core has been considered for predicting the fault proneness.

CHAPTER 2

Effectiveness of fault prediction Techniques

Chapter 2

Effectiveness of fault prediction Techniques

2.1 Introduction

Software fault prediction is helpful in deciding the amount of effort needed for software development. In literature it is observed that, a good number of approaches have been studied and evaluated on software products to determine best suitable approach for fault prediction based on certain performance criteria (precision, recall, accuracy etc.). However very less significant work has been done on feasibility of fault prediction approach. In this study, a cost evaluation framework has been proposed which performs cost based analysis for misclassification of faults. Accordingly, this study focuses on investigating two important and related research questions regarding the viability of fault prediction. First, for a given project, do the developer feel that the fault prediction results useful? In case of an affirmative answer, then it is desirable to investigate as to how to choose a fault prediction technique for an overall improved performance in terms of cost effectiveness. The proposed framework is used to investigate the usefulness of various fault-prediction techniques. The investigation consisted of performance evaluation of five major fault-prediction techniques i.e, liner regression, polynomial regression, logistics regression, Naive Bayes and SVM on Eclipse JDT core. From the obtained results, it is observed that application of fault prediction models are useful for the projects with percentage of faulty modules less than a certain threshold.

2.2 RESEARCH BACKGROUND

The following sub-sections highlight on the data set being used for fault prediction. Data was normalized to obtain better accuracy and then dependent and independent variables are chosen for fault prediction.

2.2.1 Empirical data collection

Metric suites are used and defined for different goals such as fault prediction, effort estimation, re-usability and maintenance. In this study, the mostly commonly used CK metric suite [27] is used for fault prediction. The metric values of the suite were extracted using Chidamber and Kemerer Java Metrics tool (CKJM). CKJM tools extracts OO metrics by processing the byte code of compiled Java classes. In this study, NASA and PROMISE [28] datasets are used to evaluate the impact of fault-prediction techniques over the fault removal cost using proposed model (NEcost).

2.2.2 Dependent and independent variables

The goal of this study is to establish the relationship between Object-Oriented metrics and fault proneness at the class level. In this study, a fault is used as a dependent variable and each of the CK metric is an independent variable. It is intended to develop a function between fault of a class and CK metrics suite (WMC, NOC, DIT, RFC, CBO, LCOM). Fault is a function of WMC, NOC, DIT, RFC, CBO and LCOM and can be represented as shown in the following equation:

$$Faults = f(WMC, NOC, DIT, CBO, RFC, LCOM) \tag{2.1}$$

2.2.3 Case study

In this study, to analyze the effectiveness of the proposed approach, Ellipse JDT core was used as a case study.

2.2.4 Fault Data

To perform statistical analysis, bugs were collected from Promise data repository [28]. Table 2.1 shows the distribution of bugs based on the number of occurrence (in terms of percentage of class containing number of bugs) for Ellipse JDT core.

Table 2.1: Distribution of bugs in AIP version 1.6

No. of Classes	% of bugs	Number of associated bugs
791	79.33	0
138	13.84	1
31	3.10	2
15	1.50	3
8	0.80	4
2	0.20	5
4	0.40	6
3	0.30	7
3	0.30	8
2	0.20	9
997	100.00	

Ellipse JDT core contains 997number of different classes in which 79.33% of classes contain zero bugs i.e., out of 997 classes: 791 classes contains zero bugs, 13.84% of classes contain at least one bug, 3.10% of classes contain a minimum of two bugs, 1.50% of classes contain three bugs, 0.80% classes contain four bugs, 0.20% of classes contain five and nine bugs, 0.40% classes contain six bugs, 0.30% of classes contain seven and eight bugs.

2.3 Proposed work for fault prediction

The following sub-sections highlight on the various methods used for fault classification.

2.3.1 Linear Regression models

Linear regression is the commonly used statistical technique [29]. It is used to find the linear (i.e., straight-line) relationship between variables.

The Univariate linear regression is represented as:

$$Y = \beta_1 X + \beta_0 \tag{2.2}$$

where Y represent the dependent variable and X represent the independent variable. β_0, β_1 are the constant and coefficient values respectively.

In case of multivariate linear regression, the linear regression is represented as:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p \tag{2.3}$$

Where X_i represent the independent variable and Y represent the dependent variable, β_0, β_i are the constant and coefficient values respectively.

2.3.2 Polynomial regression models

Polynomial regression is the commonly used statistical technique. Polynomial models are useful in situations where the analyst knows that curvilinear effects are present in the true response function [29]. Polynomial models are also useful as approximating functions to unknown and possible very complex nonlinear relationship.

The Univariate Polynomial regression analysis is represented as:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + ... + \beta_n X^n \tag{2.4}$$

where Y is dependent variable, X is independent variable and β_0, $\beta_1 ... \beta_n$ are the constant and coefficient values respectively.

Equation 2.4 shows the Univariate Polynomial regression model for n^{th} order polynomial. In this report, second order polynomial is considered for finding the relationship between fault and CK metrics of the class. The second order polynomial is represented as:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 \tag{2.5}$$

In case of multivariate second order Polynomial regression analysis, the Polynomial regression of two variables is based on:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_{11} X_1^2 + \beta_{22} X_2^2 + \beta_{12} X_1 X_2 \tag{2.6}$$

2.3.3 Logistic regression model

Logistic regression is the commonly used statistical technique. Which is a kind of regression analysis used for predicting the outcome of dependent variable based on one or more independent variables [13] . 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 having at least one bug. Logistic regression model is used to construct a prediction model for the fault proneness of classes. In this method, metrics are used in combination. The logistic regression model is based on the following equation:

$$logit[\pi(x)] = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_m X_m \tag{2.7}$$

where x_i represent the independent variable and $logit[\pi(x)]$ represent the dependent variable. It shows that logistic regression analysis is a standard linear regression model and the dichotomous outcome in result is transformed by the *logit* transform. This transform changes the range of $\pi(x)$ from 0 to 1 to $-\infty$ to $+\infty$, as being used for linear regression. m represents the number of independent variables. π represents the probability of fault in the class during validation. It is defined as:

$$\pi(x) = \frac{e^{\beta_0+\beta_1 X_1+\beta_2 X_2+.....+\beta_m X_m}}{1 + e^{\beta_0+\beta_1 X_1+\beta_2 X_2+.....+\beta_m X_m}} \tag{2.8}$$

2.3.4 Naive Bayes model

Naive Bayes is one of the approach for design the classifier. It is a simple probabilistic classifier which are based on applying Bayes' theorem with strong independence assumptions. A more descriptive term for the underlying probability model would be "independent feature model".

Naive Bayes classifier also called a Bayesian classification and it is based on Bayes' theorem. It assumes that all the features are independent and will not influence the estimation process. Naive Bayes classifier assigns the given object x to class $e^* = argmax_d P(d|x)$ by using $Bayes'$ rule given below:

$$P(d|x) = \frac{P(x|d)P(d)}{P(x)} \tag{2.9}$$

where $P(d)$, is the prior probability of a parameter c before having seen the data. $P(d|x)$ is called the likelihood. It is the probability of the data x and defined as

$$P(x|d) = \prod_{l=1}^{m} P(x_l|d) \tag{2.10}$$

2.3.5 Support Vector Machine model

SVM is one of the supervised machine learning model which is generally used for classification and regression analysis. SVM model analyzes data and recognizes the patterns involved in the data set [29]. SVM model acts as a non-probabilistic binary linear classifier by categorizing input data into same category or the other.

SVM is generally used for minimizing the generalization error (true error) on unseen example based on Structural Risk Minimization principle. The basic form of SVM classifier, deals with two-class problems, in which data are separated by the optimal hyperplane defined by a number of support vectors. Support vectors are the subset of the training set which define the boundary values between two classes.

The general characteristics of SVM are:

- Generalizes high dimensional spaces using small training samples.
- Obtains global optimum solution.
- Model non-linear functional relations.

The main goal of SVM is to design a model which predicts target value of the dataset in the testing phase. Thus SVM acts as a good candidate to design a model in predicting fault prone modules. The general form of SVM function is defined as:

$$Y^{'} = w * \phi(x) + b \tag{2.11}$$

where $\phi(x)$ is non linear transform. The main goal of this study is to calculate the value of w and b, so the value of $Y^{'}$ can be found by minimization of regression

risk.

$$R_{reg}(Y^{'}) = C * \sum_{i=0}^{l} \gamma(Y_i^{'} - Y_i) + \frac{1}{2} * \|w\|^2 \tag{2.12}$$

where γ represent the cost function,constant value C represents penalties for estimation error (large value of C means that errors are heavily penalized whereas a small value of C means that errors are lightly penalized). A heavier penalty trains the regression to minimize errors by making fewer generalizations. The value of w can be defined in form of data points as:

$$w = \sum_{j=1}^{l} (\alpha_j - \alpha_j^*)\phi(x_j) \tag{2.13}$$

where α and α^* represents the Lagrange multipliers , whose value is always greater and equal to zero i.e., α, $\alpha^* \geq 0$. So Equation 2.11 is modified as:

$$\begin{aligned} Y^{'} &= \sum_{j=1}^{l} (\alpha_j - \alpha_j^*)\phi(x_j) * \phi(x) + b \\ &= \sum_{j=1}^{l} (\alpha_j - \alpha_j^*) * K(x_j, x) + b \end{aligned} \tag{2.14}$$

where $K(x_j, x)$ is the kernel function, that enables the dot product to be performed in high-dimensional feature space using low-dimensional space data. In literature linear, polynomial and radial basic function used as a kernel. in this study polynomial function is used as kernel function.

2.4 Cost analysis model

This section describes the construction of a cost evaluation model, which accounts for realistic cost required to remove a fault and computes the estimated fault removal cost for a specific fault prediction technique based on the concept proposed by Wagner. Certain constraints are assumed in designing this cost evaluation model, which are as follows:

i. Different phases of testing account for varying fault removal cost.

ii. It is not practically possible to perform unit testing on all modules.

Normalized fault removal cost approach suggested by Wagner *et al.,* [30] has been used to formulate the proposed cost evaluation model. Since different projects are developed on varying platforms and in varying organization standards, the cost varies. The normalized fault removal costs are summarized in Table 2.2.

Table 2.2: Removal costs of test techniques (in staff hour per defects)

Type	Min	Max	Mean	Median
Unit	1.5	6	3.46	2.5
System	2.82	20	8.37	6.2
Field	3.9	66.6	27.24	27

The fault identification efficiencies for different testing phases are taken from the study of Jones [31]. The efficiencies of testing phases are summarized in Table 2.3. Wilde *et al* [32] have stated that more than fifty percent of modules are usually very small in size, hence performing unit testing on these modules may not be helpful.

Table 2.3: Fault identification efficiencies of different test phase

Type	Min	Max	Median
Unit	0.1	0.5	0.25
System	0.25	0.5	0.65

Equation 2.15 shows the proposed cost evaluation model to estimate the overall fault removal cost. Equation 2.16 shows the minimum fault removal cost without the use of fault prediction. Normalized fault removal cost and its interpretation is shown in Equation 2.17.

$$\begin{aligned} Ecost = {} & C_i + C_u * (FP + TP) \\ & + \delta_s * C_s * (FN + (1 - \delta_u) * TP) \\ & + (1 - \delta_s) * C_f * (FN + (1 - \delta_u) * TP) \end{aligned} \quad (2.15)$$

$$\begin{aligned} Tcost = {} & M_p * C_u * TC \\ & + \delta_s * C_s * (1 - \delta_u) * FC \\ & + (1 - \delta_s) * C_f * (1 - \delta_u) * FC \end{aligned} \quad (2.16)$$

$$NEcost = \frac{Ecost}{Tcost} = \begin{cases} < 1, & Fault\ Prediction \\ & is\ useful \\ & \\ => 1, & Perform \\ & Testing \end{cases} \tag{2.17}$$

where, Ecost represents for Estimated fault removal cost of the software when fault prediction is performed. TCost is the Estimated fault removal cost of the software without using fault prediction approach. NEcost represents the Normalized Estimated fault removal cost of the software when fault prediction is utilized.

The other notations in this cost evaluation analysis are C_i: Initial setup cost of used fault-prediction technique, C_u: Normalized fault removal cost in unit testing, C_s: Normalized fault removal cost in system testing, C_f: Normalized fault removal cost in testing, M_p : percentage of classes unit tested, FP : Number of false positive, FN : Number of false negative, TP : Number of true positive, TN : Number of true negative, TC : Total number of classes, FC : Total number of faulty classes, δ_u : Fault identification efficiency of unit testing, δ_s: Fault identification efficiency of system testing.

2.5 Experimental study

In this section, the experimental study done to find the effectiveness of fault prediction techniques for the cost based evaluation framework is presented. In this study, five techniques such as linear regression, polynomial regression, logistic regression, navie bayes, and support vector machine are used to find the classification accuracy. These five techniques is employed on Ellipse JDT core from PROMISE data repository.

2.5.1 Experiment execution

In this experiment, the values tabulated in Table 2.3 have been used in design of cost evaluation model. δ_u and δ_s show the fault identification efficiency of unit testing and system testing, respectively. The values of δ_u and δ_s have been collected

from the survey report "Software Quality in 2010" of Caper Jones [31]. M_p shows the fraction of modules unit tested, obtained from the paper of Wilde [32]. The objective is to provide the bench marks to approximate the overall fault removal cost. This is clear from the proposed cost evaluation model that if a technique is having high false negatives and/or high false positive, then it results in higher fault removal cost. When this approximated cost exceeds the unit testing cost (Tcost), it is cost effective to test all the modules at unit level instead of using fault prediction.

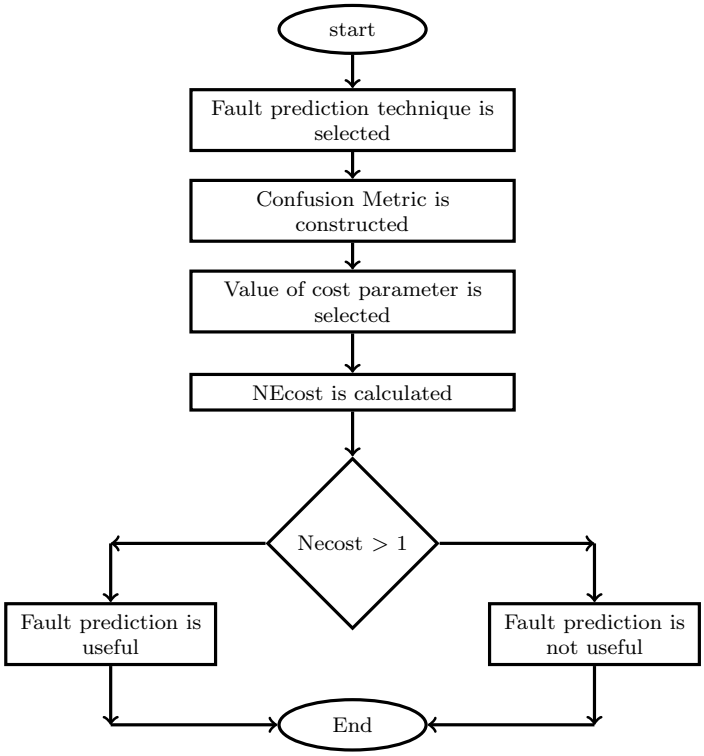

Figure 2.1: Decision chart representation to evaluate the estimated NEcost

2.5.2 Result and Analysis

In this section, the relationship between value of metrics and the fault found in a class is determined. The comparative study involves using six CK metrics as input nodes and the output is the achieved fault prediction rate. Figure 2.1 shows the flow chart for the proposed cost based evaluation framework.

Table 2.4 to Table 2.8 show the classification matrix for jdt data set for the

applied techniques such as linear regression technique, polynomial regression technique, logistic regression technique, navies bayes classification and SVM method.

- From Table 2.4, it can be observed that in case of linear regression technique, total number of 843 (747+96) classes were classified correctly with 84.55% accuracy rate.
- From Table 2.5, it can be observed that in case of polynomial regression technique, total number of 838 (749+89) classes were classified correctly with 84.05% accuracy rate.
- From Table 2.6, it can be observed that in case of logistic regression technique, total number of 840 (770+70) classes were classified correctly with 84.25% accuracy rate.
- From Table 2.7, it can be observed that in case of navies Bayes technique, total number of 835 (767+68) classes were classified correctly with 83.75% accuracy rate.
- From Table 2.8, it can be observed that in case of SVM technique, total number of 848 (769+79) classes were classified correctly with 85.06% accuracy rate.

Table 2.4: After applying Linear Regression

	Non-Faulty	Faulty
Non-Faulty	725	66
Faulty	99	107

Table 2.5: After applying Polynomial Regression

	Non-Faulty	Faulty
Non-Faulty	767	24
Faulty	148	58

Table 2.6: After applying Logistic Regression

	Non-Faulty	Faulty
Non-Faulty	771	20
Faulty	141	65

Table 2.7: After applying Naive Bayes

	Non-Faulty	Faulty
Non-Faulty	767	24
Faulty	146	60

Table 2.8: After applying SVM

	Non-Faulty	Faulty
Non-Faulty	791	0
Faulty	186	20

Table 2.9, lists the values of obtained performance parameters for Ellipse JDT core data set for the applied techniques. From Table 2.9, it can inferred that:

- Logistics regression technique obtained promising classification rate when compared to other four techniques, and also
- It can be concluded that NEcost was less than 1 for the jdt data set for all the five techniques. Logistic regression incurred negligibly less NEcost in comparison to other techniques.

Table 2.9: Result of experiment for Eclipse JDT Core

Technique	Specification	Recall	Precision	F-Measure	Accuracy	NEcost
Linear regression	0.9166	0.8799	0.6185	0.8978	83.45	0.8943
Polynomial regression	0.9697	0.8383	0.7073	0.8992	82.75	0.8879
Logistics regression	0.9747	0.8454	0.7647	0.9055	83.85	0.8823
Naives Bayes	0.9697	0.8401	0.7143	0.9002	82.95	0.8871
SVM	1	0.8096	1	0.8948	81.34	0.8886

2.6 Summary

Prediction models are used to classify fault prone classes as faulty or not faulty, but less significance has been given on the usefulness of fault prediction, which is the need of the day for researchers as well as practitioners. So cost based measures related to fault prediction needs to be modeled. In this chapter, five different prediction techniques i.e., linear regression, polynomial regression, logistic regression, Naive Bayes and SVM were applied for fault prediction. Also a note on whether using these techniques for fault prediction is useful or not in terms of cost measure was presented.

The implementation process is carried out for a case study of Ellipse JDT core. The results are generated using MATLAB. Here normalized data set of CK metrics suite was used as requisite input to the prediction models. In this study, the results suggest that, fault prediction can be useful for the projects with percentage of faulty module less than certain threshold .

CHAPTER 3

Neural Network for Fault Prediction

Introduction

Reserch Background

Proposed work

Results

Summary

Chapter 3

Neural Network for Fault Prediction

3.1 Introduction

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. In literature, prediction models are mostly developed using statistical models. Neural networks (NN) have seen an explosion of interest over the years, and are being successfully applied across a range of problem domains. Indeed, anywhere that there are problems of classification and prediction, neural networks are being used, Neural network can be used as a prediction model because it enables modeling of complex functions. In this study, artificial neural network (ANN) with Gradient Descent and Levenberg Marquardt (LM) learning methods have been used to design a classifier to classify a class as faulty or not faulty. Chidamber and Kemerer (CK) metrics suite has been considered to provide requisite input data to design the model. A case study of Eclipse JDT core has been considered for predicting a comparative study of performances of three approaches. Fault prediction is found to be useful where normalized estimated fault removal cost (NEcost) was less than certain threshold value.

3.2 RESEARCH BACKGROUND

The following sub-sections highlight on the data set being used for fault prediction. Data was normalized to obtain better accuracy and then dependent and independent variables are chosen for fault prediction.

3.2.1 Empirical data collection

Metric suites are used and defined for different goals such as fault prediction, effort estimation, re-usability and maintenance. In this study, the mostly commonly used CK metric suite [27] is used for fault prediction. The metric values of the suite were extracted using CKJM tool. In this study, NASA and PROMISE [28] datasets are used to evaluate the impact of fault-prediction techniques over the fault removal cost using proposed model (NEcost).

3.2.2 Dependent and independent variables

The goal of this study is to establish the relationship between Object-Oriented metrics and fault proneness at the class level. In this study, a fault is used as a dependent variable and each of the CK metric is an independent variable. It is intended to develop a function between fault of a class and CK metrics suite (WMC, NOC, DIT, RFC, CBO, LCOM). Fault is a function of WMC, NOC, DIT, RFC, CBO and LCOM and can be represented as shown in the following equation:

$$Faults = f(WMC, NOC, DIT, CBO, RFC, LCOM) \tag{3.1}$$

3.2.3 Case study

In this study, to analyze the effectiveness of the proposed approach, Ellipse JDT core was used as a case study.

3.2.4 Fault Data

To perform statistical analysis, bugs were collected from Promise data repository [28]. Table 2.1 shows the distribution of bugs based on the number of occurrence

(in terms of percentage of class containing number of bugs) for Ellipse JDT core.

3.2.5 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. [13]. Basili et al. studied Object-Oriented systems written in C++ language. They used the same CK metric suite. Logistic regression technique was employed to analyze the relationship between metrics and the fault proneness of classes

Table 3.1: Descriptive Statistics of Classes

Basili et al [13].	WMC	DIT	NOC	CBO	RFC	LCOM
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
Meadian	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
Eclipse JDT core	WMC	DIT	NOC	CBO	RFC	LCOM
Max.	1680	8	26	156	2603	81003
Min.	0.00	1	0.00	0.00	0.00	0.00
Meadian	20	2	0.00	7.00	30	28
Mean	58.38	2.72	0.7121	12.21	76.87	364.72
Std Dev.	135.72	1.72	2.15	17.81	180.97	3230.1

The obtained CK metric values of Ellipse JDT core are compared with the results of Basili *et al.* [13]. In comparison with Basili *et al.* the total number of classes considered is much greater i.e., 997 classes were considered (Vs. 180 as used by Basili *et al.*). Table 3.1 shows the statistical analysis of Basili et al project and Ellipse JDT core for CK Metric indicating Max, Min, Median and Standard deviation.

From Table 3.1, minimum values are almost same. But the maximum values are changes i.e., in Basili et al [13]. Maximum value of WMC is 99 but in our study, Maximum value is 1680. From Table 3.1, it is clear that the DIT metric has low value of mean and median for Eclipse JDT core. The low value of mean and median for DIT shows that inheritance was not consider much in both software system.

Table 3.2: Correlations between the Metrics of Basili et al (lower) and Ellipse JDT core (upper)

	WMC	DIT	NOC	CBO	RFC	LCOM
WMC	1.00	-0.123	0.084	0.602	0.8750	0.5123
DIT	0.02	1.00	-0.051	-0.111	-0.099	-0.055
NOC	0.24	0.00	1.00	0.2753	0.0765	0.0128
CBO	0.00	0.00	0.00	1.00	0.6133	0.39
RFC	0.13	0.00	0.00	0.31	1.00	0.6642
LCOM	0.38	0.00	0.00	0.01	0.09	1.00

The dependency between metrics is computed using *Pearson's* correlations (r: Coefficient of correlation) for Ellipse JDT core. The coefficient of correlation, r, is useful because it measures the strength and direction of the linear relationship between two variables. It is defined as the covariance of the variables divided by the product of their standard deviations. It also measures that allows us to determine how certain one can be in making predictions from a certain model. Table 3.2 shows the *Pearson's* correlation analysis for the dataset. The upper triangular matrix represents the correlations between the metrics in the Ellipse JDT core data set, and the lower triangular matrix represents the correlations between the metrics in the Basili *et al* use data sets. Correlation obtained between WMC and RFC is 0.8750 which is highly correlated i.e., these two metrics are very much linearly dependent on each other, and correlation between NOC and RFC is 0.0765 which indicates that they are loosely correlated i.e., there is low dependency between these two metrics.

3.3 Proposed work for fault prediction

The following sub-sections highlight on the various neural network methods used for fault classification.

3.3.1 Data normalization

Input feature values were normalized over the range [0,1], so as to adjust the defined range of input feature value and avoid the saturation of neurons. In literature, techniques such as Min-Mx normalization, Z-Score normalization and

Decimal scaling are available for normalizing the data. In this study Min-Max normalization [33] technique has been used to normalize the data.

Min-Max normalization performs a linear transformation on the original data. It maps each of the actual data x_i of attribute X to normalized value x_i' which lies in the range of [0,1]. The Min-Max normalization is calculated by using equation:

$$Normalized(x_i) = x_i' = \frac{x_i - min(X)}{max(X) - min(X)} \tag{3.2}$$

where max(X) and min(X) represent the maximum and minimum value of the attribute X respectively.

3.3.2 Artificial neural network (ANN) model

ANN is used for solving problems such as classification and estimation [34]. In this study, ANN is used for design the model for predicting software fault using software metrics.

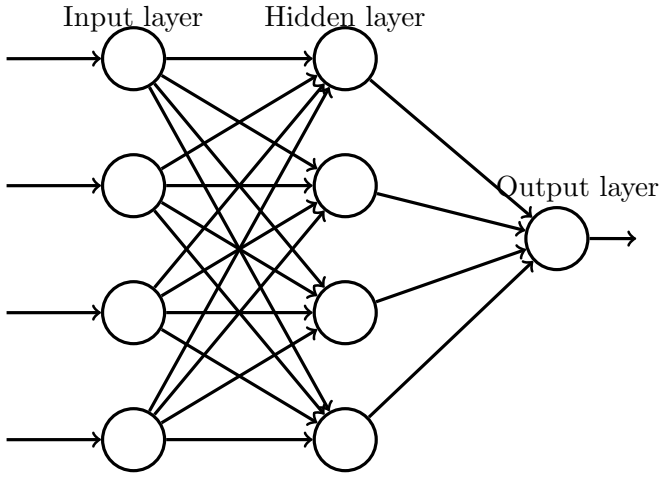

Figure 3.1: Artificial neural network

Figure 3.1 shows the architecture of ANN, which contains three layers i.e., input layer, hidden layer and output layer. Here, for input layer, linear activation function is used i.e., the output of the input layer is treated as input of the input layer. It is represented as:

$$O_i = I_i \tag{3.3}$$

For hidden and output layer, sigmoidal function or squashed-S function is used. The output of hidden layer 'O'_h for input of hidden layer 'I'_h is represented as:

$$O_h = \frac{1}{1 + e^{-I_h}} \tag{3.4}$$

and output of the output layer 'O'_o for the input of the output layer 'O'_i is represented as:

$$O_o = \frac{1}{1 + e^{-O_i}} \tag{3.5}$$

Neural network can be represented as:

$$Y' = f(W, X) \tag{3.6}$$

where Y$'$ is the output vector, X is the input vector, and W is the weight vector. The weight vector W is updated in every iteration so as to reduce Mean Square Error (MSE). MSE is based on:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y'_i - y_i)^2 \tag{3.7}$$

where y is the actual output and y$'$ is the expected output.

Different methods are available in literature to update weight vector 'W' such as: Gradient descent, Newton's method, Quasi-Newton method, Gauss Newton conjugate-gradient method and Levenberg Marquardt method etc. In this study, Gradient descent and Levenberg Marquardt are used for updating the weights vector W.

Gradient descent method

Gradient descent is one of the method for updating the weights during learning phase [35]. 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 1st order derivative of error function E_k and error function is represented as:

$$E_k = \frac{1}{2}(y'_k - y_k)^2 \tag{3.8}$$

Gradient vector G is given as:

$$G = \frac{\mathrm{d}}{\mathrm{d}W}(E_k) = \frac{\mathrm{d}}{\mathrm{d}W}(\frac{1}{2}(y'_k - y_k)^2) \tag{3.9}$$

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

$$W_{k+1} = W_k - \alpha G_k \tag{3.10}$$

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

Levenberg Marquardt (LM) method

Levenberg Marquardt method locates the minimum of multivariate function in an iterative manner. It is expressed as the sum of squares of non-linear real-valued functions [36]. This method is used for updating the weights during learning phase. It is fast and stable in terms of its executions as it is a combination of steepest descent and Gauss Newton method. In Levenberg Marquardt the weights vector W is updated as:

$$W_{k+1} = W_k - (J_k^T J_k + \mu I)^{-1} J_k e_k \tag{3.11}$$

where W_{k+1} is the updated weights, W_k is the current weights, I is the identity or unit matrix, J is the Jacobian matrix and μ is always positive, called combination coefficient i.e., when μ is very small it act as a Gauss Newton method and if μ is very large then it as a Gradient descent method.

Jacobian matrix is represented as:

$$J = \begin{bmatrix} \frac{\mathrm{d}}{\mathrm{d}W_1}(E_{1,1}) & \frac{\mathrm{d}}{\mathrm{d}W_2}(E_{1,1}) & \cdots & \frac{\mathrm{d}}{\mathrm{d}W_N}(E_{1,1}) \\ \frac{\mathrm{d}}{\mathrm{d}W_1}(E_{1,2}) & \frac{\mathrm{d}}{\mathrm{d}W_2}(E_{1,2}) & \cdots & \frac{\mathrm{d}}{\mathrm{d}W_N}(E_{1,2}) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\mathrm{d}}{\mathrm{d}W_1}(E_{P,M}) & \frac{\mathrm{d}}{\mathrm{d}W_2}(E_{P,M}) & \cdots & \frac{\mathrm{d}}{\mathrm{d}W_N}(E_{P,M}) \end{bmatrix}$$

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

3.4 RESULTS

In this section, the relationship between value of metrics and the fault found in a class is determined i.e., in all AI techniques six CK metrics are considered as input nodes and the output is the fault in the software.

The following steps are followed to design a classifier to predict faulty and non-faulty module in the software:

Step 1. Data Collection:

Data is extracted from Promise data repository.

Step 2. Normalized the dataset

Normalize the dataset over the range [0,1] using Min-Max normalization [Equation 3.2].

Step 3. Division of dataset into categories

Input data is divided into three categories i.e. training, validation and test set.

Step 4. Model design

The model is designed considering input dataset and output dataset.

Step 5. Training of network and updating Weights

Training data set is fed into the model to train the network and weights are updated using learning algorithm.

Step 6. Error calculation

Check the performance of the model. If satisfactory then stop, else again go to Step 5, update the weights and then proceed.

Step 7. Validation

Trained model will be validated by giving the validation set data.

Step 8. Testing

Finally the model is tested by feeding test set data.

3.4.1 Artificial Neural Network

ANN is an interconnected group of nodes. In this study, 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. In this study six CK metrics are taken as input, and output is the fault prediction accuracy rate. The network is trained using Gradient descent method and Levenberg Marquardt method.

Gradient descent method

Gradient descent method is used for updating the weights using Equation 3.9 and 3.10. Table 3.3 to Table 3.4 show the classification matrix for jdt data set before and after applying ANN.

Table 3.3: Before applying ANN

	Not-Faulty	Faulty
Not-Faulty	791	0
Faulty	206	0

Table 3.4: After applying ANN

	Not-Faulty	Faulty
Not-Faulty	790	1
Faulty	261	45

From Table 3.3 it is clear that before applying the logistic regression analysis, a total number of 791 classes contained zero bugs and 206 classes contained at least one bug. But after applying ANN with gradient descent method learning method, total number of 790+45 classes are classified correctly with accuracy of 83.75%.

Figure 3.2 shows the graph plot for variation of mean square error values against no. of epoch (iteration).

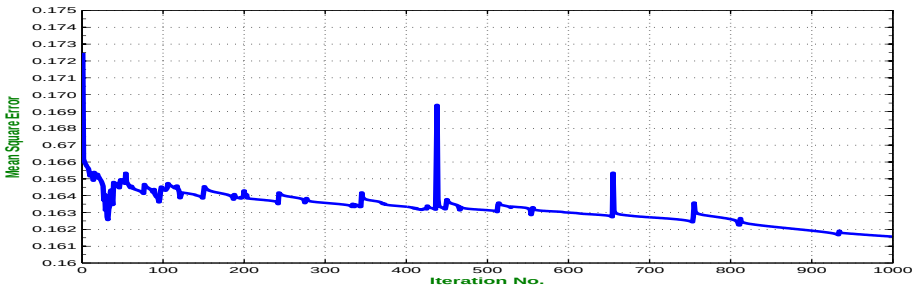

Figure 3.2: Mean square error Vs No. of iteration

Levenberg marquardt method

Levenberg marquardt method is the 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 and it consume less iteration to train the network. Table 3.3 to Table 3.5 show the classification matrix for eclipse jdt core data set before and after applying ANN with Levenberg marquardt learning method.

Table 3.5: After applying ANN

	Not-Faulty	Faulty
Not-Faulty	741	50
Faulty	142	64

From Table 3.3 it is clear that before applying the logistic regression analysis, a total number of 791 classes contained zero bugs and 206 classes contained at

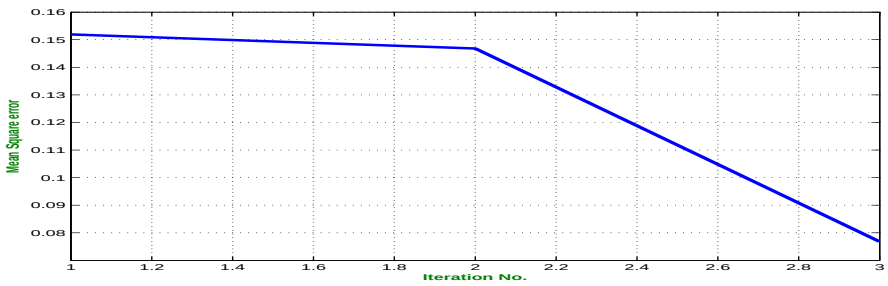

Figure 3.3: Mean square error Vs No. of iteration

least one bug. But after applying ANN with gradient descent method learning method, total number of 741+64 classes are classified correctly with accuracy of 80.74%. Figure 3.3 shows the graph plot for variation of mean square error values against no. of epoch or iteration. LM consumed only 3 iterations when compared with Gradient descent method which took 1000 iterations.

3.4.2 Fault removal cost evaluation

In this cost evaluation model, the data set of Eclipse JDT core from PROMISE repository was considered to evaluate the impact of fault prediction technique over the fault removal cost using the proposed model for computing NEcost. To illustrate effectiveness of our model, ANN classification techniques is considered. The goal is to demonstrate the cost evaluation model and suggest whether fault prediction using particular prediction technique is useful or not rather than identifying the "best" fault-prediction technique. Figure 2.1 shows the block diagram for cost evaluation model.

Table 3.6 shows the various parameters related to cost evaluation model along with NEcost. NEcost is the evaluation criteria used in evaluating a prediction techniques usefulness in fault prediction. From Table 3.6, it is evident that RBFN approach took less fault removal cost (0.8952) when compared with other approaches. From the result obtained using cost evaluation model it can be suggested that the selection of a fault-prediction technique does not only depend on the accuracy rate but it should also take into account the economics (fault removal cost) of software. However, while developing business critical applications, where ignoring faults can be crucial, then using fault prediction is not applicable; if it has high false negatives. Otherwise, it may result in a poor quality outcome.

Cost based evaluation model proposed in this study, answers two main questions which are as follows:

i. Which fault prediction model is suitable among the applied techniques.

ii. For a given software product, whether performing fault prediction analysis is economically effective or not.

Table 3.6: Fault removal cost for Eclipse JDT core

	Specification	Recall	Precision	F-Measure	Accuracy	NEcost
ANN (GD)	0.9987	0.8307	0.9783	0.9070	83.75	0.8785
ANN (LM)	0.9368	0.8392	0.5614	0.8853	80.74	0.9024

3.5 Summary

System analyst use prediction models to classify fault prone class as faulty or not faulty, which is the need of the day for researchers as well as practitioners. So more reliable approaches for prediction needs to be modeled. In this study, ANN was applied for fault prediction. The application of 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.

Fault classification using these approaches was carried out for the Eclipse JDT core case study, by coding in MATLAB environment. ANN with gradient descent learning method obtained better fault classification when compared with the Levenberg Marquardt (LM) learning method. So from the proposed work of cost evaluation model, it can be noted that it is better to perform fault prediction when NEcost is less than one.

CHAPTER 4

Hybrid ANN for Fault Prediction

Introduction

Reserch Background

Proposed work

Results

Summary

Chapter 4

Hybrid ANN for Fault Prediction

4.1 Introduction

Estimation of different parameters for Object-Oriented systems such as effort, quality and risk is of major concern in software development life cycle. Majority of the approaches available in literature for estimation and classification are based on statistical analysis and neural network techniques. Also it is perceive that numerous software metrics are consider as input for estimation. In this work, Chidamber and Kemerer metrics suite has been considered as a input data to design the classifier for classifying faulty and non-faulty module. Three artificial intelligence (AI) techniques such as: hybrid approach of neural network and genetics algorithm (Neuro-GA and adaptive Neuro-GA), hybrid approach of neural network and Particle Swarm Optimization (Neuro-PSO and Modified Neuro-PSO) and hybrid approach of neural network and Clonal Selection Algorithm (Neuro-CSA) are used for fault classification. A case study of Eclipse JDT core has been considered for predicting a comparative study of performances of three approaches. Fault prediction is found to be useful where normalized estimated fault removal cost (NEcost) was less than certain threshold value. It is observed from the obtained results that, Adaptive Neuro-GA model obtained promising results in terms of cost analysis when compared with other techniques.

4.2 Research background

The following sub-sections highlight on 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.

4.2.1 Empirical data collection

Metric suites are used and defined for different goals such as fault prediction, effort estimation, re-usability and maintenance. In this study, the most commonly used metric i.e., CK metric suite [27] is used for fault prediction. The metric values of the suite are extracted using CKJM tool. This tool is being used to extract metric values for Eclipse JDT core available in the Promise data repository [28]. The CK metric values of the Eclipse JDT core are used for fault prediction.

4.2.2 Dependent and independent variables

The goal of this study is to establish the relationship between Object-Oriented metrics and fault proneness at the class level. In this study, a fault is used as a dependent variable and each of the CK metric is an independent variable. It is intended to develop a function between fault of a class and CK metrics suite (WMC, NOC, DIT, RFC, CBO, LCOM). Fault is a function of WMC, NOC, DIT, RFC, CBO and LCOM and can be represented as shown in the following equation:

$$Faults = f(WMC, NOC, DIT, CBO, RFC, LCOM) \tag{4.1}$$

4.3 Proposed work for fault prediction

The following sub-sections highlight on the various machine learning methods used for fault classification.

4.3.1 Neural Network (NN) Model

NN are simplified models of the biological nervous system. NN is inspired by the examination of central nervous systems. Warren *et al.* in 1943 created a

computational model for neural networks based on mathematics and algorithms [34]. This computational features involved in NN architecture can be very well applied for fault prediction.

Neuro-GA Approach

Neuro-GA is a hybrid approach of ANN and GA [37]. In this approach, genetic algorithm is used for updating the weight during learning phase. A neural network with a configuration of 'l-m-n' is considered for estimation i.e., the network consists of 'l' number of input neurons, 'm' number of hidden neurons, and 'n' number of output neurons. The number of weights N required for this network can be computed using the following equation:

$$N = (l + n) * m \tag{4.2}$$

with each weight (gene) being a real number and assuming the number of digits (gene length) in weights to be d. The length of the chromosome L is computed using the following equation:

$$L = N * d = (l + n) * m * d \tag{4.3}$$

For determining the fitness value of each chromosome, weights are extracted from each chromosome using the following equation:

$$W_k = \begin{cases} \text{if } 0 <= x_{kd+1} < 5 \\ \\ -\frac{x_{kd+2}*10^{d-2}+x_{kd+3}*10^{d-3}+....+x_{(k+1)d}}{10^{d-2}} \\ \\ \text{if } 5 <= x_{kd+1} <= 9 \\ \\ +\frac{x_{kd+2}*10^{d-2}+x_{kd+3}*10^{d-3}+....+x_{(k+1)d}}{10^{d-2}} \end{cases} \tag{4.4}$$

The fitness values of each chromosome is determined based on the derived fitness function. The algorithm for deriving fitness function is as follows:

Let $(\bar{I}_i, \bar{T}_i)$; i=1,2,3....,N where

$\bar{I}_i = (I_{1i}, I_{2i}, I_{3i}, ..., I_{li})$ and $\bar{T}_i = (T_{1i}, T_{2i}, T_{3i}, ..., T_{ni})$

represent the respective input and output pairs of the neural network with a configuration of l-m-n. For each chromosome $C_i, i = 1, 2, 3, ..., p$, belonging to the current population P_i whose size is P. The following algorithm indicates the steps to find the fitness value of the individual chromosomes in the population.

Algorithm for fitness function: FITGEN()

Input: $\bar{I}_i = (I_{1i}, I_{2i}, I_{3i}, ..., I_{li})$

Output: $\bar{T}_i = (T_{1i}, T_{2i}, T_{3i}, ..., T_{ni})$

where $\bar{I}_i$, $\bar{T}_i$ represent the input and output pairs of the l-m-n configuration of neural network.

Step 1: Weights $\bar{W}_i$ from C_i are calculated using equation 4.4.

Step 2: Considering $\bar{W}_i$ as a constant weight, the network is trained for N input instances and the estimate value O_i is found.

Step 3: Error E_j for each input instance j is computed using following equation:

$$E_j = (T_{ji} - O_{ji})^2 \tag{4.5}$$

Step 4: Root mean square error (RMSE) for the chromosome C_i is computed using the following equation:

$$E_i = \sqrt{\frac{\sum_{j=1}^{j=N} E_j}{N}} \tag{4.6}$$

where N is the total number of training data set.

Step 5: Fitness value for chromosome C_i using the following equation is found out as:

$$\boxed{F_i = \frac{1}{E_i} = \frac{1}{\sqrt{\frac{\sum_{j=1}^{j=N} E_j}{N}}}} \tag{4.7}$$

Figure 4.1 shows the block diagram for Neuro-GA approach, which represent the steps followed to design the model.

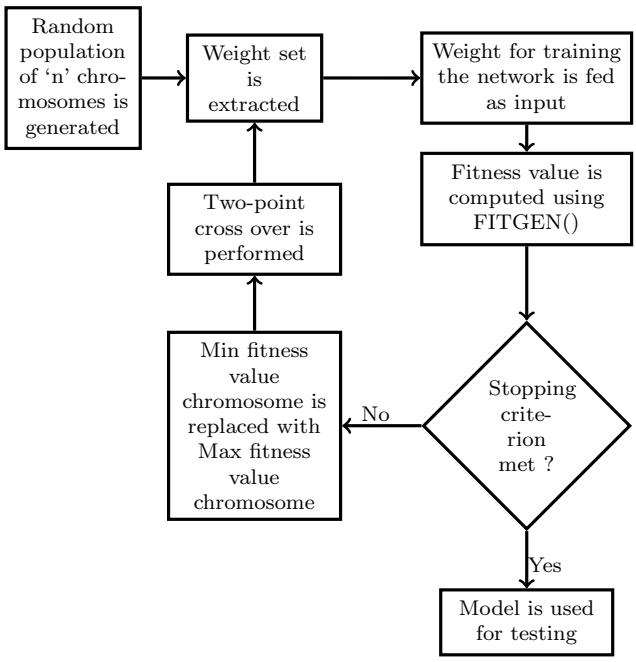

Figure 4.1: Flow chart representing Neuro-GA execution

Adaptive Neuro-GA Approach

To overcome the limitation of genetics algorithm such as premature convergence due to local optima and low convergence speed, an attempt has been made towards the improvement of parameter such as cross over probability (P_c) and mutation probability (P_m). In this study (P_c) and (P_m) values are adaptively decreased to prevent disruption of very good solution. (P_c) and (P_m) values are updated using Equation 4.8 and Equation 4.9. After implementation, it was observed that Adaptive Neuro-GA Approach gave better result in comparison of Neuro-GA.

$$(P_c)_{k+1} = (P_c)_k - \frac{C_1 * n}{5} \tag{4.8}$$

$$(P_m)_{k+1} = (P_m)_k - \frac{C_2 * n}{5} \tag{4.9}$$

where $(P_c)_{k+1}$ and $(P_m)_{k+1}$ are the updated probability of cross over and mutation, $(P_c)_k$ and $(P_m)_k$ are the current probability of cross over and mutation, C_1 and C_2 are positive constant and n is the number of chromosome having same fitness value.

Neuro-PSO Approach

Neuro Particle Swarm Optimization is a hybrid approach of neural network and Particle Swarm Optimization. In this approach, PSO is used for updating the weight during learning phase. PSO is a population based search algorithm. It is developed to simulate the behavior of birds in search for food on a cornfield [38] . In this study PSO is used as back propagation algorithm to train the network. PSO encodes the parameters of neural networks as particles and the population of particles are referred as swarm. Here, the synaptic weights of the neural network are initialized as particles and the PSO is applied to obtained the optimized set of synaptic weights. In NPSO, initially particle swarm is generated with random velocity (V) and position (X) and their fitness value is calculated using Equation 4.10.

$$\boxed{F_i = \frac{1}{E_i} = \frac{1}{\sqrt{\frac{\sum_{j=1}^{j=N} E_j}{N}}}} \tag{4.10}$$

Velocity (V) and position (x) of particles are updated using Equation 4.11 and Equation 4.12.

$$V_{k+1}^i = V_k^i + C_1 * R_1 * (Pbest_k^i - X_k^i) + C_2 * R_2 * (Gbest_k^n - X_k^i) \tag{4.11}$$

$$X_{k+1}^i = X_k^i + V_{k+1}^i \tag{4.12}$$

where

- V_{k+1}^i and X_{k+1}^i are the updated velocity and position.
- V_k^i and X_k^i are the current velocity and position.
- **Pbest** and **Gbest** are the local and global best position.
- C_1 and C_2 are positive constant, usually in between one to four.
- R_1 and R_2 are two random function whose values lies in between zero to one.

Figure 4.2 shows the block diagram for NPSO approach, which represent the steps followed to design the model.

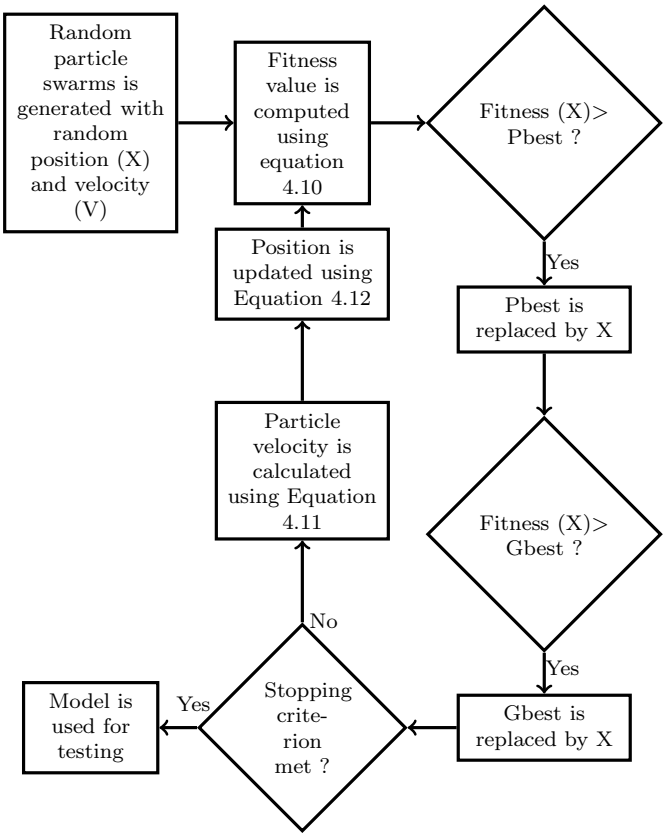

Figure 4.2: Flow chart representing Neuro-PSO execution

Modified Neuro-PSO Approach

In Modified Particle Swarm Optimization (MPSO) Approach training is same as the Particle Swarm Optimization (MPSO) Approach, but a mutation phase is incorporated just before the completion of one generation. In this study (P_m) value is adaptively decreased to prevent disruption of very good solution. (P_m) value is updated using Equation 4.13.

$$(P_m)_{k+1} = (P_m)_k - \frac{C * n}{10} \tag{4.13}$$

where $(P_m)_{k+1}$ is the updated probability of mutation, $(P_m)_k$ is the current probability of mutation, and n is the generation number.

Neuro Clonal Selection Algorithm (NCSA) Approach

Neuro Clonal Selection Algorithm (NCSA) is a hybrid approach of neural network and Clonal Selection Algorithm (CSA). In this approach, CSA is used for updating the weight during learning phase. Some possible candidate solutions are generated, antibodies will be used in the purpose function to calculate their affinity and affinity will determine the which antibody will be cloned for the next step. Cloned and mutated antibodies with a predefined ratio. After cloning and mutating, the affinity value of modified antibodies are recalculated. After certain evaluations of affinity, affinity with the smallest value is the solution closest to our problem.

The affinity values of each antibody is determined based on the derived affinity function. The algorithm for deriving affinity function is as follows:

Let $(\bar{I}_i, \bar{T}_i)$; i=1,2,3....,N where

$\bar{I}_i = (I_{1i}, I_{2i}, I_{3i}, ..., I_{li})$ and $\bar{T}_i = (T_{1i}, T_{2i}, T_{3i}, ..., T_{ni})$

represent the respective input and output pairs of the neural network with a configuration l-m-n. For each antibodies $C_i, i = 1, 2, 3, ..., p$ belonging to the current population P_i whose size is P. The following algorithm indicates the steps to find the affinity value of the individual antibody in the population.

Algorithm for affinity function: AFFINITY()

Input: $\bar{I}_i = (I_{1i}, I_{2i}, I_{3i}, ..., I_{li})$

Output: $\bar{T}_i = (T_{1i}, T_{2i}, T_{3i}, ..., T_{ni})$

where $\bar{I}_i$, $\bar{I}_i$ represent the input and output pairs of the l-m-n configuration of neural network.

Step 1: Calculate weight corresponding to individual antibody.

Step 2: Considering $\bar{W}_i$ as a constant weight, the network is trained for N input instances and the estimate value O_i is found.

Step 3: Error E_j for each input instance j is computed using following equation:

$$E_j = (T_{ji} - O_{ji})^2 \tag{4.14}$$

Step 4: Root mean square error (RMSE) for the antibody C_i is computed

using the following equation:

$$E_i = \sqrt{\frac{\sum_{j=1}^{j=N} E_j}{N}} \tag{4.15}$$

where N is the total number of training data set.

Step 5: Calculate affinity value for antibody C_i using the following equation:

$$\boxed{F_i = \frac{1}{E_i} = \frac{1}{\sqrt{\frac{\sum_{j=1}^{j=N} E_j}{N}}}} \tag{4.16}$$

Figure 4.3 shows the block diagram for Neuro-CSA approach. This block diagram represents the steps followed to design the model using Neuro-CSA approach.

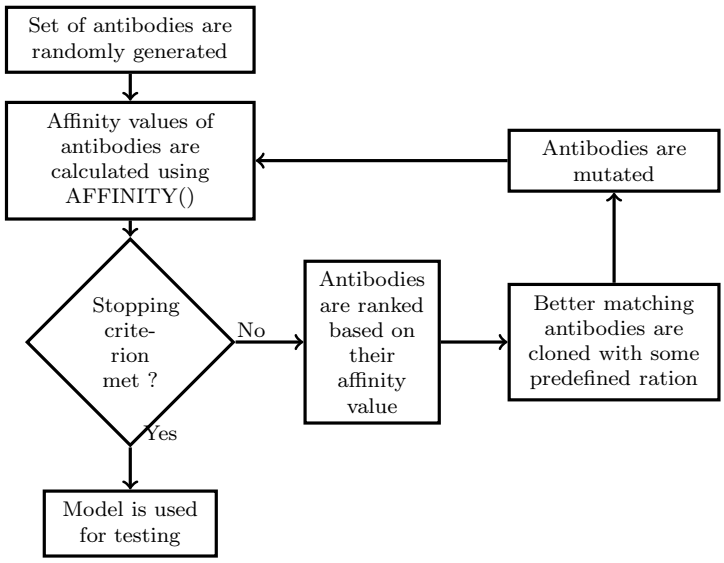

Figure 4.3: Flow chart representing Neuro-CSA execution

4.4 Result 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 Eclipse JDT core.

4.4.1 Neuro-GA Approach

Experimental setup

In this study, three layers of neural network are considered, in which six nodes act as input nodes, nine nodes represent the hidden nodes and one node acts as output node. Following are the numerical values used in execution of Neuro-GA approach for fault prediction.

i. Initialization of chromosome: Let the population of size N=50 is considered, initially generated by random process.

ii. Extraction of weight: Each chromosome contains the weight of input to hidden node and hidden node to output. Weight is extracted using Equation 4.4.

iii. Computing fitness value: The fitness of individual chromosomes is found using the proposed algorithm *FITGEN*. This algorithm is executed with an aim of minimizing the mean square error.

iv. Ranking of chromosomes: The chromosomes in the pool are ranked based on their fitness value. Minimum fitness value chromosome is stripped of by Maximum fitness value chromosome.

v. Crossover:

   - Neuro-GA: Two point cross-over approach is considered for offspring re-selection.
   - Adaptive Neuro-GA: P_c and P_m values were adaptively varied when intermediate criteria were met. while performing the simulation, the following assumption were made:
     P_c=P_c-0.1*n/10;
     P_m=P_m-0.01*n/10
     Initially P_c and P_m was taken as 0.6 and 0.1 respectively. n is no of chromosome having same fitness value.

vi. Stopping criteria: The execution of the proposed algorithm terminates when 95% of the chromosomes in the pool obtain unique fitness value, beyond this level the fitness value of chromosome get almost saturated.

In this study, Step function is used as a output function i.e, class is classified as faulty when output is greater then zero else classified as not faulty. initially 50 chromosomes are randomly generated. The input-hidden layer and hidden-output layer weights of the network are computed using equation 4.4. Two-point cross-over operation is performed in Neuro-GA and P_c and P_m values were adaptively varied in Adaptive Neuro-GA on the generated population. The execution of the algorithm converges when 95% of the chromosomes achieve same fitness values or reach maximum iteration limit (of 200 epochs). Figure 4.4 shows the variance of number of chromosomes having same fitness value and generation number for Neuro-GA and Adaptive Neuro-GA. From Figure 4.4, it is clear that Neuro-GA consume more no iteration to train the model.

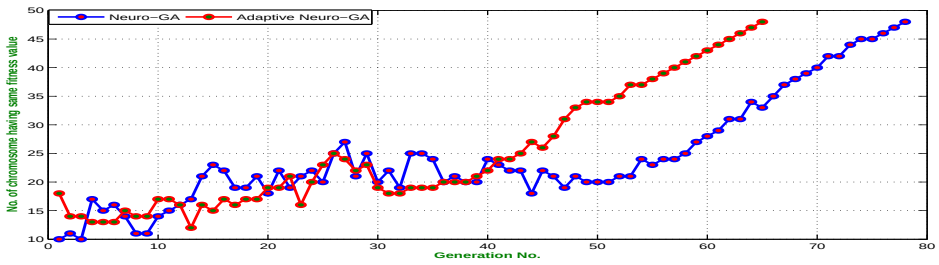

Figure 4.4: Generation No. vs number of chromose having same fitness value

Table 4.1 represent the confusion matrix for number of classes with faults before applying Neuro-GA and Adaptive Neuro-GA analysis respectively for Ellipse JDT core. Table 4.2 and Table 4.3 shows the the confusion matrix for number of classes with faults after applying Neuro-GA and Adaptive Neuro-GA analysis respectively for Ellipse JDT core.

Table 4.1: Before applying regression

	Non-Faulty	Faulty
Non-Faulty	791	0
Faulty	206	0

Table 4.2: After applying Neuro-GA

	Non-Faulty	Faulty
Non-Faulty	762	29
Faulty	62	144

Table 4.3: After applying Adaptive Neuro-GA

	Non-Faulty	Faulty
Non-Faulty	782	9
Faulty	43	163

From Table 4.2, in case of Neuro-GA total number of 762+144 classes are classified correctly with accuracy of 90.87%. From Table 4.3, in case of Adaptive Neuro-GA total number of 782+163 classes are classified correctly with accuracy of 94.78%.

4.4.2 Neuro-PSO Approach

Experimental setup

In this study, three layers of neural network are considered, in which six nodes act as input nodes, nine nodes represent the hidden nodes and one node acts as output node. Following are the numerical values used in execution of Neuro-PSO approach for fault prediction.

i. Initialization of Swarms: Let the population of size N=50 particles is considered with random velocity and position, initially generated by random process .

ii. Computing fitness value: The fitness of individual particle is found using the Equation 4.10.

iii. Comparing with Pbest: The particles in the swarm are compared with local best fitness value (Pbest), if it is better then local fitness value then Pbest is stripped of fitness value of particle.

iv. Comparing with Gbest: The particles in the swarm are compared with global

best fitness value (Gbest), if it is better then global fitness value then Gbest is stripped of fitness value of particle.

v. Mutation:

- Neuro-PSO: No any mutation step is performed.
- Modified Neuro-PSO: P_m values were adaptively varied when intermediate criteria were met. while performing the simulation, the following assumption were made:
  P_m=P_m-0.01*n/10
  Initially P_m was taken as 0.2 and n represent the generation number.

vi. Updating velocity and position: velocity (V) and position (P) of particles are updated using Equation 4.11 and Equation 4.12.

vii. Stopping criteria: The execution of the proposed algorithm continue up to 100 generation .

vi. Stopping criteria: The execution of the proposed algorithm terminates when 95% of the antibodies in the pool obtain unique affinity value.

In this study, Step function is used as a output function i.e, class is classified as faulty when output is greater then zero else classified as not faulty. initially 50 .

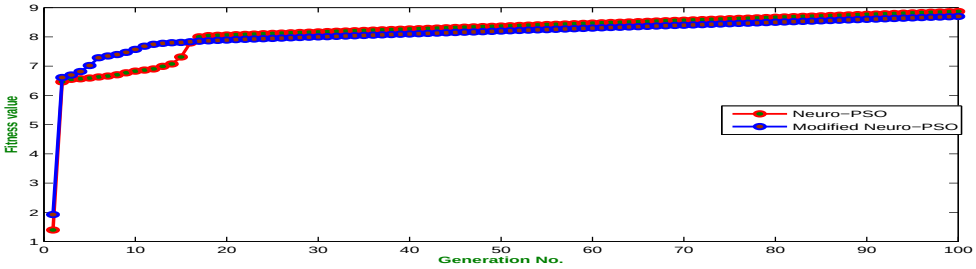

Figure 4.5: Generation No. vs fitness value

Table 4.4 and Table 4.4 shows the the confusion matrix for number of classes with faults after applying Neuro-PSO and Modified Neuro-PSO analysis respectively for Ellipse JDT core.

Table 4.4: After applying Neuro-PSO

	Non-Faulty	Faulty
Non-Faulty	781	10
Faulty	56	150

Table 4.5: After applying Modified Neuro-PSO

	Non-Faulty	Faulty
Non-Faulty	778	13
Faulty	49	157

From Table 4.4, in case of Neuro-PSO total number of 781+150 classes are classified correctly with accuracy of 93.38%. From Table 4.3, in case of Modified Neuro-PSO total number of 778+157 classes are classified correctly with accuracy of 93.78%.

4.4.3 Neuro-CSA Approach

Experimental setup

In this study, three layers of neural network are considered, in which six nodes act as input nodes, nine nodes represent the hidden nodes and one node acts as output node. Following are the numerical values used in execution of Neuro-CSA approach for fault prediction.

i. Initialization of Antibodies: Let the population of size N=50 antibodies are considered.

ii. Computing affinity value: The affinity of individual antibody is found using the Equation 4.16.

iii. Ranking of antibodies: The antibody in the pool are ranked based on their affinity value.

iv. Clone : Better matching antibodies are cloned with some predefined ratio.

v. Mutated: Mutation of antibodies are performed.

In this study, Step function is used as a output function i.e, class is classified as faulty when output is greater then zero else classified as not faulty. initially 50 antibodies are randomly generated. The execution of the algorithm converges when 95% of the antibodies achieve same affinity values or reach maximum iteration limit (of 200 epochs). Figure 4.6 shows the variance of number of antibodies having same fitness value and generation number for Neuro-CSA.

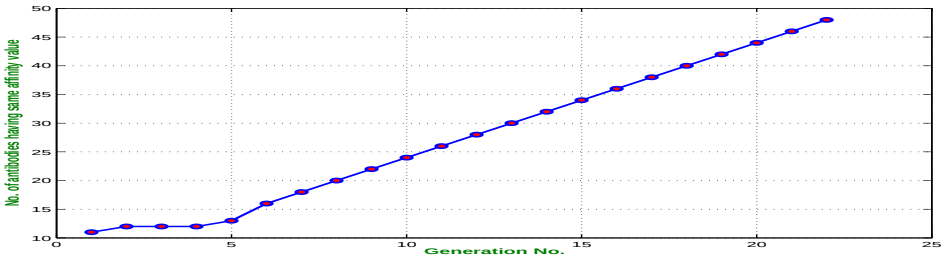

Figure 4.6: Generation No. vs fitness value

Table 4.6 show the the confusion matrix for number of classes with faults after applying Neuro-CSA for Ellipse JDT core.

Table 4.6: After applying Neuro-CSA

	Non-Faulty	Faulty
Non-Faulty	781	10
Faulty	38	168

From Table 4.4, in case of Neuro-CSA total number of 781+168 classes are classified correctly with accuracy of 95.19%.

4.4.4 Fault removal cost evaluation

In this cost evaluation model, the data set of Eclipse JDT core from PROMISE repository was considered to evaluate the impact of fault prediction technique over the fault removal cost using the proposed model for computing NEcost. To illustrate effectiveness of our model, Statistical (logistic) and machine learning (Neuro-PSO, Modified Neuro-PSO and Neuro-CSA) classification techniques are considered. The goal is to demonstrate the cost evaluation model and suggest whether fault prediction using particular prediction technique is useful or not

Table 4.7: Fault removal cost for Eclipse JDT core

	Specification	Recall	Precision	F-Measure	Accuracy	NEcost
Neuro-GA	0.9633	0.9248	0.8324	0.9437	90.87	0.8540
Adaptive Neuro-GA	0.9886	0.9479	0.9477	0.9678	94.78	0.8326
Neuro-PSO	90.9874	0.9331	0.9375	0.9595	93.38	0.8389
Modified Neuro-PSO	0.9836	0.9407	0.9235	0.9617	93.78	0.8379
Neuro-CSA	0.9799	0.9536	0.9130	0.9666	94.62	0.8334

rather than identifying the "best" fault-prediction technique. Figure 2.1 shows the block diagram for cost evaluation model.

Table 4.7 shows the various parameters related to cost evaluation model along with NEcost. NEcost is the evaluation criteria used in evaluating a prediction techniques usefulness in fault prediction. From Table 4.7, it is evident that Adaptive Neuro-GA approach took less fault removal cost (0.8326) when compared with other hybrid approaches i.e., Neuro-GA (0.8540), Neuro-PSO (0.8389), Modified Neuro-PSO (0.8379) and Neuro-CSA (0.8334). From the result obtained using cost evaluation model it can be suggested that the selection of a fault-prediction technique does not only depend on the accuracy rate but it should also take into account the economics (fault removal cost) of software. However, while developing business critical applications, where ignoring faults can be crucial, then using fault prediction is not applicable; if it has high false negatives. Otherwise, it may result in a poor quality outcome.

Cost based evaluation model proposed in this study, answers two main questions which are as follows:

i. Which fault prediction model is suitable among the applied techniques.

ii. For a given software product, whether performing fault prediction analysis is economically effective or not.

4.5 Summary

In this chapter, an attempt has been made to use software metrics in order to build Object-Oriented software reliability prediction models. In this study, three hybrid approaches of soft computing viz., Neuro-GA, Neuro-PSO and Neuro-CSA were applied for fault prediction. Fault classification using these approaches were carried out for the Eclipse JDT core case study, by coding in MATLAB environment. The hybrid approach obtained better fault classification when compared with statistical and machine learning approaches. In terms of cost evaluation, it can be concluded that hybrid approach has low value of NEcost, when compared with that of statistical and machine learning approaches. So from the proposed work of cost evaluation model, it can be noted that it is better to perform fault prediction when NEcost is less than one.

CHAPTER 5

Conclusion and Future work

Chapter 5

Conclusion and Future Work

Software quality assurance process focuses on the identification and removal of faults quickly from the artifacts that are generated and subsequently used in the development of software. Software fault prediction is one of the different strategies, which are conducted rapturously during the very early stage of software development life cycle (SDLC). Fault prediction information not only points to the need for increased quality during the development but also provides an information to understand suitable verification and validation activities in order to improve the effectiveness.

The effectiveness of a fault-prediction technique is demonstrated by educating it over a part of some known fault data and measuring its performance against the other part of the fault data. Recently, several software project data repositories became publicly available such as NASA Metrics Data Program and PROMISE Data Repository. Availability of these public data sets has encouraged undertaking more rigorous investigations on their replications. Large number of fault-prediction techniques have been applied to demonstrate their effectiveness on these data set.

In this study, a cost evaluation framework has been proposed, which performs cost based analysis for misclassification of faults. Accordingly, this study focuses on investigating two important and related research questions regarding the viability of fault prediction.

- 1. For a given project, are the fault prediction results useful?

- 2. In case of an affirmative answer, then look for how to choose a fault prediction technique for an overall improved performance in terms of cost effectiveness.

The proposed framework is used to investigate the usefulness of various fault-prediction techniques. The investigation consisted of performance evaluation of different prediction techniques i.e., statistical method, machine learning method and hybrid approach of machine learning and soft computing methods i.e., Neuro-GA, Neuro-PSO and Neuro-CSA on public datasets.

From the obtained results, it is observed that fault prediction approach is useful for the projects with percentage of faulty modules less than a certain threshold. It was observed that there was no single technique that could provide the best results in all cases. However, for different critical applications, where ignoring faults can be crucial, using fault prediction may not be effective if it has high false negatives. Otherwise, it may result in a poor quality outcome.

Further, work can be carried out on another quality parameter such as : software maintainability. Software Maintainability is typically measured as change effort. Change effort can mean either the average effort to make a change to a class, or the total effort spent on changing a class.

Bibliography

[1] 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*, vol. 186, 1994.

[2] J. Bansiya and C. G. Davis, "A hierarchical model for Object-Oriented design quality assessment," *ACM Transactions on Programming Languages and Systems.*, vol. 128, pp. 4–17, August 2002.

[3] B. K. Kang and J. M. Bieman, "Cohesion and reuse in an Object-Oriented system," in *Proceedings of the ACM SIGSOFT Symposium on software reuseability*, pp. 259–262, Seattle, March 1995.

[4] L. C. Briand, J. W*üst*, J. W. Daly, and D. V. Porter, "Exploring the relationships between design measures and software quality in Object-Oriented systems," *The Journal of Systems and Software*, vol. 51, pp. 245–273, May 2000.

[5] L. Etzkorn, J. Bansiya, and C. Davis, "Design and code complexity metrics for Object-Oriented classes," *Object-Oriented Programming*, vol. 12, no. 10, pp. 35–40, 1999.

[6] M. Halstead, *Elements of Software Sciencel*. New York, USA: Elsevier Science, 1977.

[7] B. Henderson-Sellers, *Software Metrics*. UK: Prentice-Hall, 1996.

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

[9] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, pp. 308–320, December 1976.

[10] 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, pp. 241–262, 1995.

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

[12] S. R. Chidamber and C. F. Kemerer, "A metrics suite for Object-Oriented design," *IEEE Transactions on Software Engineering*, vol. 20, pp. 476–493, June 1994.

[13] 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, pp. 751–761, October 1996.

[14] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial use of metrics for Object-Oriented software: An exploratory analysis," *IEEE Transactions on Software Engineering*, vol. 24, no. 8, pp. 629–639, 1998.

[15] L. C. Briand, J. Wüst, S. V. Ikonomovski, and H. Lounis, "Investigating quality factors in object-oriented designs: an industrial case study," in *Proceedings of the 21st international conference on Software engineering*, pp. 345–354, ACM, 1999.

[16] M.-H. Tang, M.-H. Kao, and M.-H. Chen, "An empirical study on object-oriented metrics," in *Software Metrics Symposium, 1999. Proceedings. Sixth International*, pp. 242–249, IEEE, 1999.

[17] M. Cartwright and M. Shepperd, "An empirical investigation of an object-oriented software system," *IEEE Transactions on Software Engineering*, vol. 26, no. 8, pp. 786–796, 2000.

[18] K. El Emam, S. Benlarbi, N. Goel, and S. N. Rai, "The confounding effect of class size on the validity of object-oriented metrics," *IEEE Transactions on Software Engineering*, vol. 27, no. 7, pp. 630–650, 2001.

[19] R. Subramanyam and M. S. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297–310, 2003.

[20] 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.

[21] H. J. A. S. Khoshgoftaaar T.M, Allen E.B, "Application of neural networks to software quality modeling of a very large telecommunications systems," *IEEE Transactions on Neural Networks*, vol. 8, no. 4, pp. 902–909, 1997.

[22] A. E. H. J. Hochman R, Khoshgoftar T.M, "Evolutionary neural networks: a robust approach to software reliability problems," in *Proceedings of the Eight International Symposium on Software Reliability Engineering*, pp. 13–26, 1997.

[23] C. B. Jiang Y and M. Y, "Techniques for evaluating fault prediction models," *Empirical Software Engineering*, vol. 13, no. 5, pp. 561–595, 2008.

[24] M. T and K. R, "Revisiting the evaluation of defect prediction models," in *Proceedings of the 5th International Conference on Predictor Models in Software Engineering (PROMISE)*, pp. 1–10, 2009.

[25] M. T and K. R, "Effort-aware defect prediction models," in *Proceedings of 14th IEEE European Conference on Software Maintenance and Reengineering (CSMR)*, pp. 107–116, 2010.

[26] B. L. Arisholm E and J. E.B, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Empirical Software Engineering*, vol. 83, no. 1, pp. 2–17, 2010.

[27] S. R. Chidamber and C. F. Kemerer, *Towards a metrics suite for Object-Oriented design*, vol. 26. ACM, 1991.

[28] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, "The promise repository of empirical software engineering data," June 2012.

[29] Y. Zhou and H. Leung, "Predicting object-oriented software maintainability using multivariate adaptive regression splines," *Journal of Systems and Software*, vol. 80, no. 8, pp. 1349–1361, 2007.

[30] W. S, "A literature survey of the quality economics of defect-detection techniques," in *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering (ISESE)*, pp. 194–203, 2006.

[31] J. C, "Software quality in 2010: a survey of the state of the art," in *Founder and Chief Scientist Emeritus*, 2010.

[32] R. Huitt and N. Wilde, "Maintenance support for object-oriented programs," *IEEE Transactions on Software Engineering*, vol. 18, no. 12, pp. 1038–1044, 1992.

[33] J. Kaur, S. Singh, K. S. Kahlon, and P. Bassi, "Neural network-a novel technique for software effort estimation," *International Journal of Computer Theory and Engineering*, vol. 2, no. 1, pp. 17–19, 2010.

[34] W. McCulloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[35] 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.

[36] 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.

[37] C. Burgess and M.Lefley, "Can genetic programming improve software effort estimation," *Information and Software Technology*, vol. 43, pp. 863–873, 2001.

[38] J. Zhou, Z. Duan, Y. Li, J. Deng, and D. Yu, "Pso-based neural network optimization and its utilization in a boring machine," *Journal of Systems and Software*, vol. 178, no. 1, pp. 19–23, 2006.

[39] Z. Pawlak, "Rough sets," *International Journal of Computer and Information Sciences*, vol. 11, no. 5, pp. 341–356, 1982.

[40] R. S. (Ed), *Intelligent Decision Support. Handbook of Applications and Advances of the Rough Sets Theory.* Dordrecht: Kluwer Academic Publishers, 1992.

Dissemination of Work

Accepted

1. Lov Kumar, Yeresime Suresh and Santanu Ku. Rath Fault Prediction for Apache Open Source Framework using Chidamber and Kemerer Metrics Suite. *in Proceedings of CONSEG: 7th International Conference on Software Engineering, pp. 105-111, Pune, 2013.*

2. Lov Kumar and Santanu Ku. Rath A Model to Assess the Effectiveness of Fault Prediction Techniques for Quality Assurance . *Accepted in : 26th International Conference on Software Engineering and Knowledge Engineering (SEKE 2014).*

3. Yeresime Suresh, Lov Kumar and Santanu Ku. Rath. Statistical and Machine Learning Methods for Software Fault Prediction using CK Metric Suite: A Comparative Analysis. *ISRN Software Engineering, vol. 2014, pp. 1-15, Hindawi Publisher.*

4. Yeresime Suresh, Lov Kumar and Santanu Ku. Rath. A Cost Evaluation Framework for Software Fault Prediction using Radial Basis Function Network. *Accepted for publication in: International Journal of Information Processing (IJIP), vol. 8, no. 2, IK Publishers..*

Communicated

1. Lov Kumar and Santanu Ku. Rath Effectiveness of Fault-Prediction Techniques for Quality Assurances of Embedded Software. *International Conference on Harware/Software Codesign and System synthesis*, 2014.