S. I. HASSEN

ATILIM UNIVERSITY 2022

A NEW METHOD FOR SOFTWARE DEFECT PREDICTION BASED ON OPTIMIZED MACHINE LEARNING TECHNIQUES

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

ATILIM UNIVERSITY

SHAHO ISMAEL HASSEN

A MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

JANUARY 2022

A NEW METHOD FOR SOFTWARE DEFECT PREDICTION BASED ON OPTIMIZED MACHINE LEARNING TECHNIQUES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ATILIM UNIVERSITY

BY

SHAHO ISMAEL HASSEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JANUARY 2022

Approval of the Graduate School of Natural and Applied Sciences, Atilim University.

Prof.Dr. Ender KESKINKILIÇ
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of **Master of Science in Computer Engineering Atilim University**.

Assoc. Prof. Dr. Gökhan ŞENGÜL

Head of Department

This is to certify that we have read the thesis A NEW METHOD FOR SOFTWARE DEFECT PREDICTION BASED ON OPTIMIZED MACHINE LEARNING TECHNIQUES Submitted by Shaho Ismael Hassen and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science-Computer Engineering.

Prof.Dr. Alok MISHRA
Co-Supervisor

Prof.Dr. Ali YAZICI
Supervisor

Prof. Dr. Ali YAZICI
Software Eng. Department, Atılım University

Asst. Prof.Dr. Selma NAZLİOĞLU
Software Eng. Department, Atilim University,

Assoc. Prof.Dr. Ibrahim ALPER DOĞRU,
Computer Eng. Department, Gazi University,

Date: *14-01-2022*

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

SHAHO, HASSEN:

Signature:

ABSTRACT

A NEW METHOD FOR SOFTWARE DEFECT PREDICTION BASED ON OPTIMIZED MACHINE LEARNING TECHNIQUES

SHAHO HASSEN,

M.S. Computer Engineering, Atilim University, Ankara

Supervisor : Prof.Dr. Ali YAZICI

Co-Supervisor : Prof.Dr. Alok MISHRA

January 2022, 84 pages

In this thesis a novel and robust heuristic driven neuro-computing model was developed for software defect prediction. Unlike other classical machine learning models, neuro-computing, especially Levenberg Marquardt Neural Network (LM-ANN), is considered to be more robust in terms of adaptive learning, which can be vital towards non-linear feature learning and hence defect data. However, similar to the other machine learning models, the likelihood of local minima and convergence could not be avoided due to exceedingly high weight estimation for 17 input features. Considering this fact, this research contributed a novel improved genetic algorithm, say heuristic model was developed to assist ANN for adaptive weight estimation and update during learning. Here, the key purpose of heuristic model was to help LM-ANN gaining superior weight estimation, update and hence learning without undergoing any local minima and convergence problem. This as a result helped the proposed neuro-computing model to achieve higher accuracy than the classical neural network over targeted software fault datasets. In addition to the classifier or machine learning improvement, in this research the focus was made on feature engineering as well that helped alleviating any probability of class imbalance, over-fitting and convergence.

Keywords: Neuro-Computing, LM-ANN, Defect Prediction, Genetic Algorithm, ANN, Local Minima, HNC-SDP, and Levenberg Marquardt.

ÖZ

OPTİMİZE EDİLMİŞ MAKİNE ÖĞRENİM TEKNİKLERİNE DAYALI YAZILIM KUSURLARINI ÖNGÖRMEK İÇİN YENİ BİR YÖNTEM

SHAHO HASSEN,
Yüksek Lisans, Bilgisayar Mühendisliği, Atılım Üniversitesi,
Danışman: Prof. Dr. Dr. Ali YAZICI
Yardımcı Danışman: Prof. Alok MISHRA

Ocak 2022, 84 sayfa

Bu tezde, tüm gerçekleri motivasyon olarak kabul ederek yazılım kusur tahmini için yeni ve sağlam bir buluşsal güdümlü nöro-bilgisayar modeli geliştirilmiştir. Diğer klasik makine öğrenimi modellerinden farklı olarak, nöro-bilgisayar, özellikle Levenberg Marquardt Sinir Ağı (LM-YSA), doğrusal olmayan özellik öğrenimi ve dolayısıyla hatalı veriler için hayati önem taşıyabilecek uyarlamalı öğrenme açısından daha sağlam olarak kabul edilimektedir. Ancak, diğer makine öğrenimi modellerinde olduğu gibi, 17 giriş özelliği olanlarda da aşırı yüksek ağırlık tahmini nedeniyle yerel minimum ve yakınsama olasılığından kaçınılamamıştır. Bu gerçeği göz önünde bulundurarak, bu araştırma, öğrenme sırasında uyarlanabilir ağırlık tahmini ve güncelleme için YSA'ya yardımcı olamak amacıyla buluşsal model denilen yeni bir geliştirilmiş genetik algoritm sunark katkıda bulunmuştur. Burada buluşsal modelin temel amacı, LM-YSA'nın herhangi bir yerel minimum ve yakınsama sorunu yaşamadan üstün ağırlık tahmini, güncelleme ve dolayısıyla öğrenme elde etmesine yardımcı olmaktır. Sonuç olarak , önerilen nöro-bilgisayar modelinin hedeflenen yazılım hatası veri kümeleri üzerinde klasik sinir ağından daha yüksek doğruluk elde etmesine yardımcı olmuştur. Sınıflandırıcı veya makine öğrenimi iyileştirmesine ek olarak, bu araştırmada, herhangi bir sınıf dengesizliği, aşırı uydurma ve yakınsama olasılığının hafifletilmesine yardımcı olan özellik mühendisliğine de odaklanılmıştır.

Anahtar kelimeler: Nöro-Bilgisayar, LM-ANN, Kusur Tahmini, Genetik Algoritma, ANN, Local Minima, HNC-SDP, ve Levenberg Marquardt.

To my father, who left us in his body, but his soul still flutters in the sky of my life….

To the wellspring of love, altruism and generosity, my great mother….

ACKNOWLEDGMENTS

Throughout the process of writing this thesis, I was given a lot of help and encouragement.

I would first like to thank my advisor, Professor Dr. Ali Yazici, whose expertise was invaluable in formulating the research questions and methodology. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

I would also like to thank my co-advisor Professor Dr. Alok Mishra, for his valuable guidance throughout my studies. You provided me with the tools that I needed to choose the right direction and successfully complete my thesis.

Furthermore, I would like to thank the jury member for their wise counsel and sympathetic ear.

Finally, without the help of my family and my friends, I would not have been able to finish my thesis.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF SYMBOLS/ABBREVIATIONS

FOSS	Free Open-Source Software Components
SMOTE	Synthetic Minority Over Sampling
ANN	Artificial Neural networks
LM	Levenberg Marquardt
SDP	Software Defect Prediction
LOC	Line of Code
DIT	Depth of Inheritance Tree
CKJM	Chidamber and Kamrer Java Machine Driven Metrics
COQUALMO	Constructive Quality Model
HNC	Heuristically Driven Neuro-Computing
ULR	Univariate Logistic Regression

CHAPTER 1
INTRODUCTION

Over the most recent couple of many years software technologies have become inevitable need of contemporary human society; though thrust to innovate and improve at-hand solution is still the key driving force behind software industry [1][2]. On the other hand, increasing global competitiveness, decentralized and global operating culture too have forced industry to produce software solutions with minimum development cycle delay and cost [1]. In this reference, developers or enterprises have been trying to reuse free-open-source software components (FOSS) or even function-reuse concept to reduce time as well as cost [2][3]. Despite of the cost-efficiency of software reuse paradigm, the events of aging [4], smelling [4][5], fault [6][7] etc. remain the key challenge particularly due to excessive reuse of the FOSS or software components [8][9]. The exceedingly high reuse of software components often leads software faults and malfunction and hence impacts software reliability [2]-[9]. Additionally, there can be the different reasons including inter-class ambiguity, aging etc. that can eventually lead fault or system failure. Such fault probability raises question on the efficacy and reliability of such (software) solutions for varied purposes. In the past numerous events can be found which were caused due to inappropriate software design and allied failures [4][6][7][9].

The matter of fact is that there are numerous strategically important and sensitive software computing environment such as financial software, healthcare's computer aided diagnosis as well as electronic healthcare records, science and technologies, defense, and industrial monitoring and control where reliability of the software remains as uncompromisable need [9][10]. Thus, ensuring software reliability is must while preserving cost-factor or allied expectations.

To address software reliability demands, software defect detection and allied prediction have always been the vital practice [9]. However, design complexity, different development paradigms etc. make major t hand solution confined. The process of fault detection turns out to be more severe and complex in case of large software system that

often encompass thousands or even more highly complex classes or autonomous functions. Exploring in depth one can find that most of the software development firms or allied enterprises perform manual testing to detect software fault or defect; however, performing defect prediction over a large complex software solution is infeasible by means of manual process. Moreover, manual testing and detection methods often carry the probability of misdetection, or human-error that makes its suspicious.

Despite a few innovations, the industry has been applying classical manual testing methods including regression and or black-box approaches; however, it turns highly resource exhaustive and time-consuming that eventually hinders the goal of enterprise [6]. Considering it as motivation, automatic reusability prediction approaches have been proposed, which exploits the different software metrics characterizing the software design features and machine learning methods [11]-[12][13]. In this reference, though a few efforts are made by exploit software metrics information to perform detect prediction in each class (of software) or function; however, the optimality of the algorithms used particularly machine learning based classifications (say, classifiers) remained suspicious. This is because, the unavoidable nature of local minima, convergence kinds of computational limitations. Moreover, different researches advocate the different software metrics towards defect prediction; however, no significant study assesses their optimality so that a computationally efficient (software defect prediction) solution could be accomplished. This key fact has been considered as the critical main impetus behind this review.

Exploring at hand solutions, one can find that the majority of the existing machine learning driven software defect prediction (SDP) systems apply employing either code structure such as the line of code (LOC), depth of inheritance tree (DIT), often ignore the intrinsic association amongst the classes or components such as cohesion and coupling. Similarly, those approaches mainly focus on complexity aspect to perform reusability prediction with standalone classifier. On the contrary, merely employing complexity as the code-feature can’t be suitable as the coupling and cohesion too are equally important to be considered when assessing defect prediction each class. This is because the excessive reuse nature of the different classes too can give rise to the

software fault and defects [8][9][11]-[12]]. In addition to the above stated issues of the existing system, one key problem has remained unexplored so far in history, and this is class-imbalance. This is because the probability of faulty class or defect class would be significantly lower than the normal classes. Therefore, learning any classical machine learning method over such imbalanced data might force the model to show false positive (i.e., inclined towards majority class). Such probability can be more severe under local minima and pre-mature convergence condition. Hence, to mitigate such problems, alleviating class-imbalance problem followed by machine learning optimization are must. Considering this as motivation, in this theory a profoundly vigorous heuristic driven machine learning model is developed for software defect prediction. Architecturally, the proposed model incorporates data resampling using synthetic minority over sampling (SMOTE) sampling algorithm followed by Min-Max normalization and heuristic driven neuro-computing for software defect prediction.

Noticeably, the proposed software defect prediction model applies object-oriented programming (software) metrics extracted from software solutions. Thus, exploiting OOP- Chidamber and Kamrer Java machine driven metrics (often called CKJM metrics), the proposed heuristic driven neuro-computing model performs two-class classification. To evaluate execution of the proposed software defect prediction model, different NASA PROMISE datasets have been utilized for simulation, where the proposed model has exhibited superior performance in terms of accuracy, precision, recall and F-Score. The detailed discussion of the proposed model and allied corresponding results inferences are given in Chapter-3 and Chapter-4, respectively.

1.1 Initiation

For any research or study, a well-defined and structured discussion of the research subject matter, contemporary significance, research gap and allied scopes and possible contributions play decisive role. In this reference, this chapter primarily discusses some of the key theoretical discussions pertaining to software engineering, challenges in the software engineering, software design and allied challenges, software defect prediction, software metrics for defect prediction etc. In addition, this section puts glance on the research artifacts including research objectives, problem formulation,

thesis contribution and allied thesis outline. Before discussing the research contribution(s), a snippet of the key theoretical discussions including software engineering, software defect prediction, and object-oriented programming software metrics etc. are given in this section.

1.1.1 Software Engineering

Generically, software is defined as "a pre-characterized and organized figuring programs, particularly intended to play out certain targeted task(s)". The lofty institution, Institute of Electrical and Electronics Engineers (IEEE) Standard of Software Engineering Terminology has defined software engineering as the "…systematic and well calibrated employment of skilled, disciplined, and quantifiable paradigm for software development, functional monitoring, operation, and its maintenance".

Software engineering encompasses design, development and maintenance of software applications. Software engineering can be defined in different terms. Some of the generic definitions of software engineering are given as follows:

- *Software engineering represents the orderly and aligned utilization of the scientific and technological information and respective extracted knowledge, approach, and practices for the design, development, testing or maintenance and documentation of software.*
- *Software engineering represents the process of certain target-oriented research, system design, development and its functional test operating systems-level software, software related compilers, and various network distribution systems for financial utilities, businesses, medical, industrial, defence, communications, scientific, education, aerospace, and various genetic computational utilities.*

This is the matter of fact that the software engineering domain flourished to fulfil up-surging socio-economic and industrial computing and decision-making demands. However, ensuring reliability of computation often remained challenge. To meet

software demands, it is vital to ensure optimally designed, resource efficient and reliable software systems which could meet overall expectations.

The key limitation for the effective use of software in past was hypothesized to be the "software crisis". A software of project lacking effective and proper planning can possess defects and threat of fault proneness. Such fault proneness often imposes resource as well as time exhaustion to perform fault detection and removal. Surprisingly, the time and cost consumed in fault or defect identification and its removal is more than the development cost. On contrary, it can also be stated that zero defects cannot be practical, especially large-scale software and complex applications. Even though performing intensive fault detection and removal, during the course of utilization, there can be numerous faults still existing in software application. It leads towards a never-ending unpredictable scenario of the software, which sometime turns out to be catastrophic or unusable.

In general, a software solution defines certain product with defect probability by characterizing known or unknown bugs or defects, which leads towards an iterative process of developing software while ensuring the elimination of known defects and exploring or revealing new faults overtime. In this approach, since the new bugs or faults are revealed, removing or fixing detected bugs of the highest priority is the predominant duty of a software development team. There are a number of conventional approaches used to identify or detect defects in software applications. Some of the predominant approaches are the code review and analysis, functional unit testing and integrated system testing etc. In addition, it also comprises a dedicated approach for software defect removal which in general accompanies software evaluation and design processes.

In the modern world, computing devices have turned out to be the inevitable fraction of human lifestyle where it serves major purposes including telephonic communication, social media, business decisions, query-based solution, industrial computation and analytics, scientific researches etc. Noticeably, these all-aforesaid applications involve certain software programmes to execute certain task. In this case, guaranteeing reliability is of utmost significance. Towards software reliability,

different efforts have been made; however, software defect prediction systems are the predominant approaches. Functionally, these methods exploit different feature engineering and machine learning methods to identify or detect fault in software systems. A snippet of the software defect prediction terminologies and allied activities is given in the subsequent sections.

1.1.2 Software Defect Prediction

Typically, the term defect in software domain signifies programming error, functional erroneous outputs, fault, pre-mature system failure or shut-down, inaccurate computation, and misleading tasks etc. In fact, in software domain error can be based on the different tasks intended or application environment for which the software is designed. The term fault or software defect resembles as the end-outcomes for these two are same [12]. However, IEEE Standard Glossary of Software Engineering Terminology [14] characterizes error as the human (programming) mistakes that gives wrong processing outputs. Though, system failure caused because of wrong program or malicious in-system entity also can't be neglected [14]. In this thesis, the term fault or defect is related with botches at the coding level. These mix-ups are seen during programming testing at unit just as framework levels. Even though, the malicious entity identified during system testing can be stated to be failures, it remains unrelenting while employing the term default since it can be expected that all the reported malicious variables are identified at the coding level.

Majority of defects are caused due to various mistakes and errors initiated by people in certain source code or its functional design, or operating systems employed by such programs, and a few are created by compilers generating inaccurate source code. Typically, software defect prediction signifies the approach to identify the possible or existing faults within the system so as to improve overall reliability and avoid any future adversaries. In reference to the automatic defect prediction, this mechanism involves predictive classification framework that exploits programmes or code-features to identify malicious component or entity throughput the program. Unlike classical human-based manual testing methods, automatic defect detection models are

more efficient as it alleviates any probable human mistake due to tiring one-by-one analysis concept. In this reference, the machine learning based software defect prediction model can be characterized as a code-feature driven learning environment in which exploiting the software metrics or characteristics the tool identifies the probability of fault or presence of fault within the program [15]. In this thesis, in order to develop a novel defect prediction system, this philosophy has been taken into consideration where the fault data has been taken regarded as dependent variable and software metrics as the independent variable. Generally, a software defect prediction models comprises software metrics or code-characteristics as the independent variables, while shortcoming-probability as the dependent variable. This combination of variables helps designing a learning environment for defect detection over test condition. [16].

There are a number of mechanisms employed for software defect prediction. Only a few of the widely used methods are listed here used for defect prediction are regression technique, affiliation rule mining, clustering, classification, neural network, decision tree etc. Taking into consideration of the robustness of regression technique and neural network, in this thesis a hybrid model has been developed that employed multivariate regression and enhance neural network for fault detection and classification, respectively. Regression technique evaluates the correlation between variables. It investigates at how the dependent or response variable interacts with the independent or predictor variables. The connection is stated as an equation in which the response variable is predicted as a linear function of the predictor variable.

1.1.2.1 Software Defect Prediction Techniques

Software Defect Prediction techniques are employed to classify and predict faulty and non-faulty modules and number of defects supposed to be discovered in a software module. For the most part, approaches have been offered fault of defect prediction and classification. These techniques can extensively be classified into techniques employed for predicting expected number of defects to be observed in a given software

artefact (Prediction) and techniques which are employed for prediction if or not the given software artefact supposed to have a fault or defect (Classification).

Figure 1.1 represents the generic classification of defect prediction systems. Here the classification has been done based on the purpose of defect count prediction.

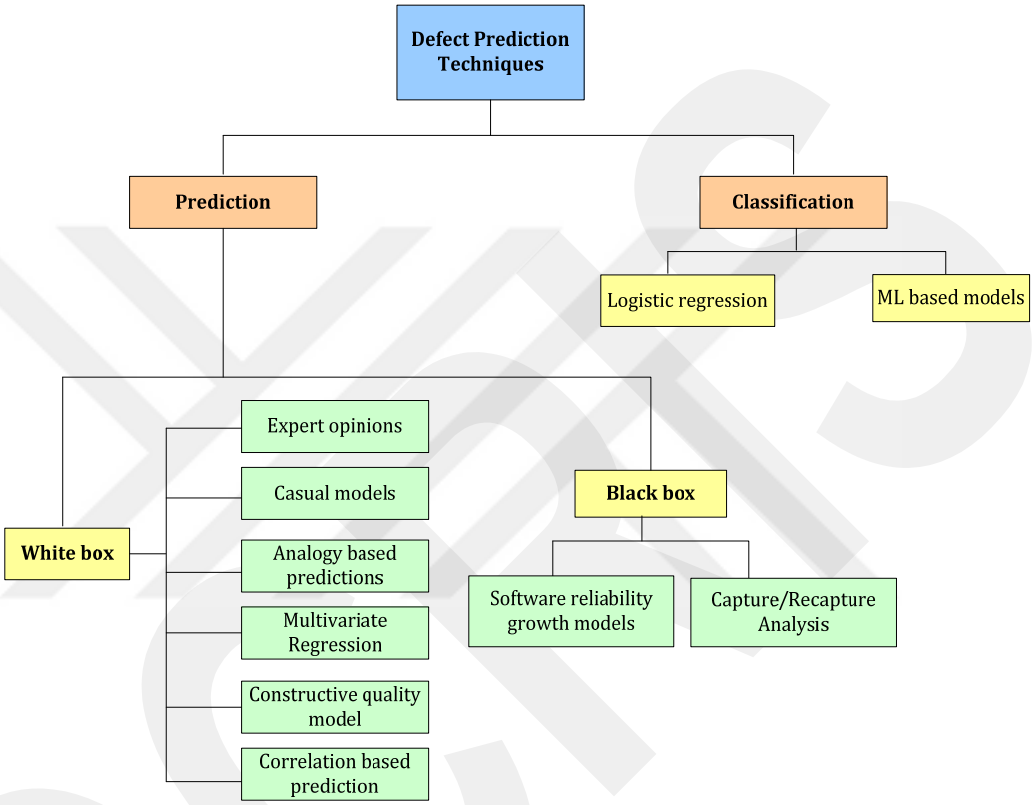

Figure 1.1 Overview of different software defect prediction techniques

A brief discussion of the different defect prediction techniques (Figure 1.1) is given in the following sections.

1.1.2.2 Techniques for Number of Defects Estimation

The prediction models might only employ the number of defects revealed during development and testing without taking into consideration of other attributes associated the internal structure/design/implementation of the project/product – they are categorized as black box defect prediction approaches. In contrast, software defect

prediction techniques that use attributes associated with the process and products such as size, complexity, changes are classified under white box techniques.

A. ***Models for Software Reliability***

Models for Software Reliability Growth represents the mathematical equations employed for modelling the growth of the software of the system reliability by means of defect inflow data from the development or the testing phase. Efficient approaches are considered based on software development or software testing or by using empirical assessment of the performance of a sub-set of frameworks on the testing data, which is then employed for selecting suitable frameworks and enable the defect prediction.

B. Capture Recapture analysis

This SDP paradigm is functional on the basis of the analysis of patterns of defects identified in a certain specified software artefact using independent fault detection processes [17]. In this approach, the latent defects count is calibrated by means of the overlap among various defects detected by independent detection activities. This approach is also known as capture recapture paradigm [18].

C. Expert opinions

In case of the availability of the experts, the swift and flexible approach of SDP is employing them for defect predictions on the basis of the experience. The predominant limitation of this paradigm is its subjective nature and ineffectiveness for scaling down efficiently at lower granularity levels. This approach can be significant in major scenarios where the defect prediction has to be accomplished at project level encompassing huge functional components, especially in that case where the available experts can design based on their previous experience to forecast defects, but when defect predictions are to be made at lower granularity levels, this approach is found to be confined.

D. Causal models

Causal models always make effort to formulate generic relationships between various software products attributes and the software processes with regard to the amount of flaws that are expected to be discovered in the software system.

E. Analogy based predictions

The estimation paradigm based on analogy predominantly depends upon the retrieval and comparison of a number of software metrics existing in between the previous project and the present one so as to recognize the majority of the analogous project [10]. Especially for SDP problems, usually size, kind of application, functional complexity and various parameters are employed to find comparable projects so that predictions may be made.

F. Multivariate Regression

The SDP models based on regression techniques used to be statistical regression which is employed for making defect predictions by means of certain set of software metrics or change in the attributes of the software codes as predictor variables. Approaches such as multiple linear regression can be taken into consideration for calculating the estimated number of bugs in a certain provided software module. Numerous software process and software product metrics can be employed as the independent variables to perform regression-based SDP process. Among these feasible metrics the metrics such as code complexity and change in source code are considered predominantly for SDP utilities.

G. Constructive quality model (COQUALMO)

The approach of the constructive quality model [19] is functional on the basis of the software defect identification and removal as formulated in [20], which is in fact analogous to the mechanism proposed in [21]. This approach employs expert estimated defect introduction, identification and removal sub-models so as to form a quality model commonly known as COQUALMO. In this approach, initially the number of non-trivial necessities, design and coding defects incorporated are

measured by means of Defect Introduction (DI) sub-model that employs the size of software and other key attributes and process. Primarily, the output of DI sub-model is employed as input to the Defect Removal (DR) component along with inputs retrieved from defect removal profile levels and software size values. In fact, the resultant of the DR sub-model is nothing else but the estimation of the number of residual defects per unit size [19].

H. Correlation based models

Correlation based approaches employs fault or the defect data retrieved while performing software development or testing. The number of defects retrieved at certain provided iteration while performing development is employed for predicting the number of defects. Researchers [22]evaluated the association between software defects observed in the earlier and later iterations by means of linear regression approach. Both these approaches; regression and correlation employ linear regression for defect count prediction.

1.1.2.3 Techniques for Defect Classification

The other predominant approach of software defect prediction is nothing else but the software defect classification, which strive for identification of fault-prone software modules by means of a number of software attributes. In general, the defect prediction models and fault classification approaches are employed at the lower granularity levels like class level or file level. Software artefacts recognized as defect prone can be prioritized so as to get more intensive verification and validation functionalities.

A. Logistic regression

In general, the logistic regression approach can be employed for classifying software modules as defective or non-defective. Using a similar strategy of multivariate regression, the software metrics range are employed as predictor variables that performs classification of the software systems.

B. Machine learning models

The SDP models based on machine learning employs statistical paradigms and data mining approaches for defect prediction and classification. These approaches are similar to the regression-based paradigms and employ independent variables as input variables. The dominant difference is that machine learning approaches used to be dynamic learning approach which intend to optimize its, as more data is made available. Noticeably, amongst the major automatic SDP methods, machine learning driven approaches have performed superior, where the use of efficient feature engineering and classifiers can improve the performance even better. This hypothesis has been regarded to be the primary driving factor behind this study. Realizing the at hand limitations of the existing systems, in this thesis a feature improved environment with heuristically driven neuro-computing model is developed for SDP. Unlike classical machine learning methods, like artificial neural network, this study proposes a robust heuristic concept, also called evolutionary computing to improve ANN learning so as to remedy the situation of local minima and convergence. Thus, with the proposed (improved) ANN model, here onwards called heuristically driven neuro-computing (HNC) machine learning model, each class software prediction is performed.

This is the matter of fact that merely improving the classifier environment can't yield optimal performance and therefore, retaining superiority of the feature to be learnt is equally important. In this reference, this study applied CKJM OOP metrics which were learnt to identify the fault probability (per-class). The detail of the considered OOP-CK metrics is given in the subsequent section.

1.1.3 OOP-CK Software Metrics

Software metrics, also called code-characteristics are the key features characterising design aspects including line of code, its complexity, coupling between or amongst classes, cohesion of development etc. [29]. Exploiting such characteristics, one can easily predict the presence of any erroneous or probable faulty component in a software system [29]. In this reference, software metrics are those software metrics which characterises the quality of the software developed in terms of aforesaid

complexity, cohesion, coupling, size, pattern, smell etc. [29]. Considering its significance, numerous efforts have been made where researchers have exploited or identified the different sets of metrics characterizing the software quality [23]. For instance, the metrics presented in Figure 1.2 depict the key object-oriented metrics which is adaptable to perform quality assessment [23].

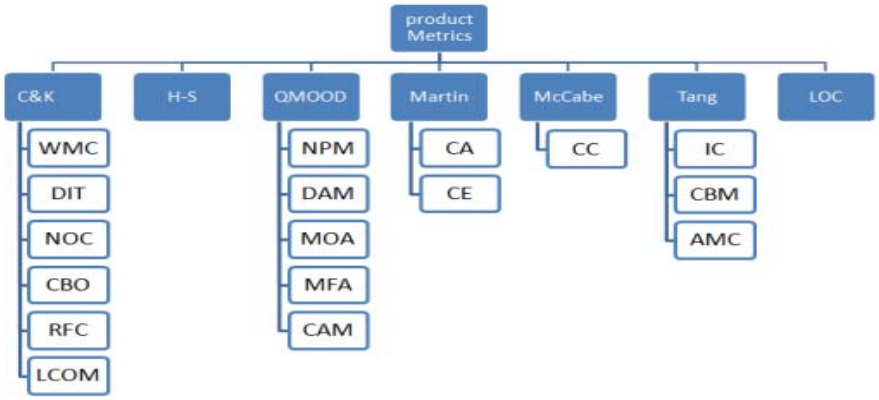

Figure 1.2 Various Metrics' Characteristics

- *WMC- Weighted Methods per Class*
- *DIT- Depth of Inheritance Tree*
- *NOC- Number of Children*
- *CBO – Coupling between Object Classes*
- *RFC- Response for a Class*
- *LCOM-Lack of Cohesion in Methods*
- *Ca- Afferent Couplings*
- *Ce- Efferent Couplings*
- *NPM- Number of Public Methods*
- *DAM- Data Access Metric*
- *MOA- Measure of Aggregation*

- *MFA- Measure of Functional Abstraction*
- *CAM- Cohesion Among Methods of Class*
- *CC-Cyclomatic Complexity*
- *LOC- Lines of Code IC- Inheritance Coupling*
- *CBM- Coupling Between Methods*
- *AMC- Average Method Complexity*

Recalling the fact that the different software feature or metrics can have the different characteristics, the feature selection methods are applied to assess how suitable a feature can be towards defect detection or prediction. In sync with this motive, feature selection performs assessing the feature subsets and its significance across the search space. This process resembles the feature assessment where one intends to assess what key features are available in a person and how significant that person is to perform certain task. However, there can be the multiple features highly in-sync with each other which can have the direct cumulative impact on prediction result. And therefore, selecting the suitable set of features towards prediction turns out to be the non-convexity complex task. Some literatures like [23]identifies the following features (Table 1.1) as suitable towards defect prediction.

Table 1.1 Type of metrics considered in study [30]

Software Metrics	Category
WMC, NOC and RFC	Object-Oriented Software Metrics (Intra-class metrics)
LCOM	Internal (hybrid) Metrics
CBO	Static Metrics (WMC, DIT)

1.2 Research Objectives

Considering overall research intends and allied scopes, in this dissertation certain objectives have been identified. These research objectives are given as follows:

- To develop a robust heuristically driven neuro-computing model (HNC) for software defect prediction.
- To perform feature resampling and Min-Max normalization approaches to improve defect prediction accuracy over OOP-CK metrics.
- To evaluate the proposed system's performance HNC-driven SDP model (say, HNC-SDP) over the different benchmark datasets when it comes to classification accuracy, precision, recall and F-Measure.

1.3 Research Questions

In sync with the above stated research objectives, this study formulates certain research questions, for which this study intends to achieve suitable answers. These questions are:

RQ1: Can OOP-CK Metrics be effective towards software defect prediction?

RQ2: Can the use of Heuristically Driven Neuro-Computing (HNC) be effective towards software defect prediction (HNC-SDP)?

RQ3: Can the use of feature engineering like resampling and Min-Max normalization help alleviating class-imbalance and over-fitting (including convergence and local minima) problem to yield superior prediction performance?

Thus, the general goal of the study is to find the best solutions to the above-mentioned problems.

1.4 Thesis Outline

The overall study and allied dissertation are divided into five distinct chapters, where each chapter discusses the individual intend or subject matter. The details of the outline considered is given as follows:

Chapter-1 Introduction

This chapter mainly discusses the introduction of the overall study, including the theoretical discussions about software engineering, reliability problem in software, software defect prediction and allied systems, software metrics towards software

defect prediction. Likewise, this part talk about the research destinations, research questions and overall thesis outline.

Chapter-2 Literature Survey

This chapter focuses on some of the significant literatures in the field to the software defect prediction. The qualities just as impediments of the diverse existing strategies are examined in this part.

Chapter-3 Method and Materials

In this chapter, the overall proposed model (i.e., HNC-SDP) model implementation and allied algorithmic discussions are given.

Chapter-4 Results and Discussion

In this chapter, the simulation findings are presented. and allied comparison with the existing methods are discussed in details.

Chapter-5 Conclusion

In this chapter, the overall research conclusion, future scopes and allied inferences are discussed. The references utilized in this review are given toward the finish of the document.

CHAPTER 2
LITERATURE SURVEY

2.1 Background

For any research exploring allied literatures or existing systems (say, algorithms) helps researcher to understand existing approaches and their strengths or limitations. In this reference, this chapter focuses on some of the significant literatures related to the various topics in software defect prediction techniques and their characteristics. The key motive of this chapter is to understand strengths, weaknesses and allied scopes for further innovation or optimization. The following parts provide a more in-depth examination of the various literatures investigated.

2.2 Literature Survey

This section highlights some of the main literatures on the various topics in machine learning methods for software defect prediction, such as data mining-based methods, regression-based methods, neuro-computing-based defect prediction methods etc.

2.2.1 Data Mining Based SDP

Liu et al. [24]explored the key issue of software quality and employed historical patterns of the associated software metrics retrieved from a software project and eventually suggested a generic multi-data training and validation model for fault classification. Researchers used NASA MDP fault datasets to determine the effectiveness of their suggested system.

Song et al. [25] in their research work focussed on defect prediction and made efforts to correct efforts using association rule mining technique where they employed their model with more than 200 projects. Researchers achieved higher accuracy in terms of defect prediction as well as its correction measures. Through their research model, they claimed 23% better accuracy as compared to C4.5, IDE based mining techniques.

Lessmann et al., [26] investigated the performance of classification algorithms with 10 NASA Metric data sets using 22 classifiers.

Munson et al., [27] in their research work explored the efficiency of discriminating analysis approach, which is a statistical based paradigm, for detect prediction and its earlier detection. They implemented principal components analysis (PCA) in order to minimize multi-collinear complexity metrics to uncorrelated measures on symmetrical intricacy areas.

Tom [28] classified fault data using Naïve Bayesian algorithm and stated that their proposed Naïve Bayesian based approach can give better results because of the conditional independence hypothesis.

Ohlsson et al. [29] processed for fault detection with telecommunication software modules. Riquelme et al.[30] employed fault datasets retrieved from PROMISE repository and extracted software metrics form the projects, which they used for defect prediction using conventional genetic algorithm Look for rules that describe subgroups that have a high chance of being faulty. Researchers just employed the conventional genetic algorithm, even for searching faults throughout datasets, which seems highly intricate and computationally complicate, especially for large scale datasets.

Catal et al., [31] developed an SDP system based on Artificial Immune System (AIS). The proposed classifier replicates the characteristics of the antigen and the antibody in the duration of faults caused due to pathogens. This model was based on human biological functioning towards pathogens caused defects and its causative factor recognition.

Drown et al. [32] made the effort to use evolutionary computing or evolutionary programming concept called evolutionary sampling, which is nothing else but GA based data sampling approach that enhances software quality assurance and its reliability. Researchers empirically found that evolutionary sampling-based scheme performs better for defect prediction.

Chen et al. [33] proposed a data mining-based software defect prediction system, and examined performance for different mining and Bayesian Network based models.

Wang et al. [34] explored defect prediction models based on C4.5 mining algorithm. They introduced the Spearman's rank correlation coefficient to select root node of the decision tree for defects prediction. Researchers compared the performance in terms of size of decision tree and observed that the further developed models decreased the size of the choice tree by 49.91% and prediction accuracy was on two modules, the rise was 4.58 percent and 4.87 percent, respectively..

Qinbao et al. [25] proposed an approach based on association rule mining technique for software defect predication and its removal. Their proposed system exhibited higher detection accuracy with minimal false-negative. Comparative results with PART, C4.5, and Naive Bayes indicated a 23 percent increase in accuracy.

Biwen et al. [35] proposed a C4.5 decision tree algorithm-based defect prediction system. The novelty of their approach was the implementation of a k-medoids algorithm to cluster functions which was followed by deriving functional scenarios, and then to use the C4.5 model to predict the fault in the scenarios.

Marwala [36] proposed a Bayesian detailed neural network technique utilizing half breed Monte Carlo to scaled form angle technique for distinguishing flaws in structures utilizing modular properties and the direction models confirmation measure and vibration information. An irregularity identification strategy for spacecraft system dependent on affiliation rules was proposed by Yairi et al., [37] . Tree-based models were utilized by Koru & Tian [38] in their research work explored into the inter-relationship between software complexity and probability of defects in software projects. Researchers examined their analytical approach with six large-scale software products from organizations like IBM and Nortel Networks.

Aside from that, data mining-based SDP systems, numerous approaches have been proposed using machine learning. The following section discusses some of them.

2.2.2 Machine Learning Driven SDP

Zhang et al., [39] in their research work suggested for machine learning algorithms for SDP applications. Authors suggested that machine learning can be a potential technique for software quality assessment, cost analysis and effort optimization etc.

Researchers in their work also found that machine learning with its robust mathematical and logical justifications can enable it to be used for major applications and hence it can be used for software defect prediction applications.

Venkata et al., [40] they developed a framework in their study that may be utilized for defect prediction, especially for the mission-critical SDP by means of a Memory-Based Reasoning (MBR) paradigm. Research facilitated an option for coordinating users by means of certain well defined logical process for selecting the appropriate MBR architecture which might facilitate the optimal prediction based on software defect data.

Trendowicz et al [41] surveyed many techniques to modelling quality and stated that a software quality model can be of great significance for software quality assurance initiative.

Khoshgoftaar [42] described six software quality estimation techniques, namely Regression Tree (RT), Fuzzy Systems (FS), Case-Based Reasoning (CBR), Rule-Based Systems (RBS), Multiple Linear Regression (MLR) and Neural Networks (NN). They found that that no any single quality model can be optimal to ensure quality optimization, rather multiple schemes can be used for optima defect prediction, Researchers illustrated the combination of fuzzy & rule-based system.

Zhong et al., [43] in their work employed an unsupervised learning approach for SDP application, where the unsupervised learning approach was implemented to construct a software quality estimation model. They used various techniques such as human expert system, clustering scheme etc for model development.

Venkata et al., [44] studied various approaches for SDP applications and focussed on machine learning, where they evaluated this technique with various NASA's MDP fault datasets. Researchers found that all the probable defects were frequently behaved abnormally in terms of skewness, non-linear and varying values of variances, and higher outlier characteristics.

Sheppered et al., [45] compared various approaches of SDP, where they used different approaches such as rule-based approach, stepwise regression, artificial neural network (ANN) and case-based reasoning.

Mair, et al., [46] explored different types of software defect prediction on real time software data. Researchers in their work compared their proposed approach for its accuracy, explanatory value, and its configurability. A similar comparative research was done in [47] where they performed comparison for varied case-based fault classification and researchers found that there can't be much significant by combining various parameter altogether. Similarly, researchers [48] explored various approaches, where some approaches such as adaptation strategies, k-NN and rule-based techniques can be the effective approaches.

Khoshgoftaar et al., [49] claimed that by removing outliers and data noise from the training data set, model accuracy may be increased.

Khan et al., [50] analysed and examined about software quality expectation approaches dependent on case-based thinking, fuzzy logic, neural networks, support vector machine, expectation maximum probability algorithm and Bayesian belief network.

Chug et al., [51] proposed different arrangement and clustering techniques to predict software defect. The performance of three data mining classifier algorithms named J48, Random Forest, and Naive Bayesian Classifier (NBC) were assessed dependent on different measures like ROC, Precision, MAE, RAE etc. It was then followed by implementation of the clustering technique on the data set using k-means, Hierarchical Clustering and Make Density Based Clustering algorithm. Evaluation of the findings for clustering was made dependent on standards like Time Taken, Cluster Instance, Number of Iterations, Incorrectly Clustered Instance and Log probability etc.

Wang et al., [52] suggested Multi- variants Gauss Naive Bayes (MvGNB) by executing prediction scheme for defect prediction and compared it with decision tree learner J48. Further, Singh et al., [53] examined the Naive-Bayes classifiers and J48 classifier for its effectiveness for defect prediction, where they integrated it with supervised discretization approach (SDA) for defect identification. Researchers exhibited that the integration of SDA technique can significantly optimize the fault prediction accuracy.

Qinbao et al., [25] developed an SDP model that facilitates fair and comprehensive performance analysis of different SDP systems. Researchers through their retrieved results affirmed that learning approaches must be taken into consideration for varied data sets.

Pushphavathi et al., [54] in their research work, proposed a hybrid approach containing random forest and Fuzzy C Means clustering approach for SDP system. At first, they employed the random forest algorithm for initial screening and ranking of variables. Consequently, they fed new dataset as the input of the FCM technique, which is responsible for forming certain interpretable architecture for defect prediction. To assess the effectiveness of the intended model, they employed a 10-fold strategy for cross-validation. Researchers employed FCM and RF algorithms to the software decisive components such as people and process.

Menzies et al., [55] exhibited that Naive Bayes with logNums filter-based approaches can be effective for software defect prediction (SDP). To legitimize the exhibition, they utilized NASA MDP datasets for SDP performance assessment.

Shuai et al., [56] proposed dynamic Support Vector Machine (SVM) algorithm the basis of a novel cost-sensitive SVM (CSSVM) technique, which was later improved using GA. Researchers used the precision of geometric categorization as the fitness value of the genetic algorithm so s to enhance the performance of the proposed CSSVM algorithm.

Challagulla et al., [44] in their research work explored a number of different techniques for software defect prediction. Some of the predominant approaches evaluated by researchers are stepwise multi-linear regression technique for SDP, multivariate analysis-based SDP model, ANN and instance-based reasoning approaches for machine learning based SDP techniques and Bayesian-belief networks-based approaches. In addition, they also explored the efficiency of decision trees algorithms for software quality prediction.

Ardil et al., [57] assessed the efficacy of feed forward ANN towards prediction-oriented learning. On the other hand, Jianhong, et al., [58] applied different ANN

variants including Resilient Back propagation algorithm towards software defect prediction, and found that the backpropagation method can give superior results.

Yuan, et al., [59] applied adaptive network based Fuzzy-model (ANFIS) for software defect prediction. This approach exploited the reliability metrics as independent variable to perform training. However, the overall process was too exhaustive with no benchmark justification over a software with imbalanced fault class.

Bezerra, et al., [60] applied Probabilistic Outputs based RBF (RBF-DDA) for software defect detection. Authors found that ANN variant performs superior over k-NN machine learning method. Yet, authors failed in addressing the suitability assessment of the different metrics.

Menzies et al., [61] applied J48 machine learning method to perform software defect prediction over NASA defect dataset. They found that Naïve bayes methods are superior over J48 and linear regression methods.

Fujimakiet al., [62] exploited system behaviour information to perform software defect prediction.

Koru & Liu [38] on the other hand assessed relative performance of J48, KStar, ANN, Bayesian networks, and SVM algorithms, where they found that J48 exhibits superior accuracy over the other state-of-art methods.

Mende & Koschke [63] assessed the effect of irregular forests algorithms according to artificial immune systems with the region beneath the receiver operating qualities (ROC) curve utilized as a performance assessment measure. They observed that irregular forests accomplishing the most noteworthy exactness.

Kanmani et al, [64] employed two neural network methods and found that the neuro-computing method with better non-linear feature learning can be more powerful towards SDP.

Researchers in [14] applied Bayesian belief networks (BBN) for defect software prediction. A comparable exertion was made in [65] also where authors found it superior over classical machine learning approaches.

Shatnawi et al., [46] found that Chidamber and Kamerer object-oriented metrics might be a more appropriate way to describe software towards DP. In this referencing, citing the defect data of NASA-MDP [66]authors found that ANN can be a suitable learning model to perform defect prediction. Interestingly, authors stated that CBO, WMC, RFC, and SLOC can be the optimal set of features towards defect prediction.

Malhotra et al., [67] too found that ANN can be a superior machine learning environment towards defect prediction and found is better over the existing boosting, Naive bayes algorithms for defect prediction. Janes et al., [68] utilized models, for example, Poisson regression and zero-inflated negative binomial regression for real-time, telecommunication systems along with machine learning algorithms.

Tomaszewskiet al., [69]in their research work exhibited that measurable strategies are superior to master assessments.

Because this thesis is based on the idea of the object-oriented SDP has been already carried out, as a result, a review of existing methodologies using software metrics inevitably occurs. The literature covering various strategies employing is discussed in the next section, object-oriented software metrics (CK Metrics) be debated.

2.2.3 Software Metrics planted SDP

Taking into consideration of the significance of software metrics and its implementation for software quality assessment, it is significant to discuss various researches conducted for its evolution and its implementation for applications such as defect prediction etc. Some of the predominant software metrics do represent software design metrics, code metrics, and various other types of metrics.

CK-Metrics [23], especially WMC, NOC, DIT, CBO, RFC, and LCOM were found more significant and informative to perform software defect prediction. A snippet of these metrics is given in the previous chapter 9Chapter-1). Authors [83] assessed the inter-relationship amongst the different features to understand their cumulative impact examined the efficacy of defect prediction and discovered that the usage of CK metrics can be vital towards defect prediction.

Subramanyam et al., [70] applied WMC, CBO, and DIT as software metrics software.

Nagappanet et al. [71] assessed defect by employing OOP-metrics. Authors agreed that the use of object-oriented metrics can provide more intrinsic information to perform defect prediction. Systa [72] exploited UML diagrams and allied graph structure to perform defect prediction. Authors stated that the UML can be vital towards both static as well as dynamic feature characterization and hence can help in performing defect identification.

Schroter et al., [73] applied reverse engineering paradigm to exploit OOP-metrics for defect prediction. Authors found that exploiting inter-class relation the presence of defects can be assessed or investigated. Similarly, Ostrand et al., [74] stated that assessing any modifications in the code can help identifying the probability of fault. They too hailed that the use of code-component relationship can help performing defect prediction.

Nagappan et al. [75] found that code components and allied intrinsic features can be vital towards defect identification. In this reference, El-Emam et al. [47] stated that WMC and code size has the direct impact on the code smell and vulnerability.

Briand et al., [76] in their research work examined 49 software products for defect class identification, prediction and classification. Researchers employed both the univariate as well as multivariate analysis scheme for individual faulty class identification. They developed a multi-agent model and observed that NOC metric are not mush significant for defect prediction. Yu et al., [77] observed majority of software metrics are significant for fault prediction except DIT.

Xu et al., [66] assessed object-oriented CK metrics for defects evaluation using statistical analysis and neuro-fuzzy techniques. The findings revealed the importance of trustworthiness defect assessment can be accomplished utilizing metrics SLOC, WMC, CBO and RFC metrics.

Zhou & Leung [78] used the logistic regression and machine learning methods for fault prediction using CK metrics suite.

Olague et al. [79] examined software quality by exploiting CK metrics, MOOD and QMOOD, and revealed that CKJM metrics are more accurate and informative towards machine learning-based defect (automatic) prediction.

Alshayeb et al. [80] too stated the same inference and found that System Design Instability (SDI) metrics, in addition to the aforesaid CK metrics are vital towards defect analysis and prediction. Harrison et al. [81]alternatively, noted that the utilization of statistical models such as logistic regression can be vital to understand intrinsic CK features and their association. This as a result can help performing defect perdition. However, authors could not justify their efficacy over the other state-of-art methods.

Basili et al., [82] explored into the software design metrics with an intension to analyse the errors proneness and its probability of inclusion at different phases of software development. Cartwright et al. [25] exploited OOP-CK metrics extracted from a large software comprising 133,000 LOC. Exploiting the intrinsic features, authors revealed that in large scale software the different features such as inheritance, polymorphism etc. cannot be effective as standalone feature for further classification and hence CK metrics seems to be the potential approach for automatic (machine learning-based) software defect prediction [83]. The similar inference was given by Moser et al. [83] who applied Java Eclipse software fault data to perform defect identification.

2.3 Summary

This section addresses some of the most important works on the domain of software defect prediction. Different machine learning methods including decision tree, regression, neuro-computing etc. were examined for their respective performance towards defect prediction. Amongst the major solutions, neuro-computing were found more effective, yet no existing approach could address the class-imbalance problem followed by convergence and local minima issues in the classical machine learning. Noticeably, the depth assessment of the survey also shown that the employment of evolutionary computing algorithms like genetic algorithms, AIS etc. can be vital to improve data. However, these algorithms have not been applied to enhance the

performance of any machine learning methods, for defect prediction. In sync with this observation, in this dissertation a novel heuristic driven neuro-computing environment is proposed for software defect prediction, where OOP-CK metrics are used as benchmark datasets.

CHAPTER 3
METHOD AND MATERIALS

3.1 Background

This section primarily discusses the overall proposed model (i.e., heuristic driven neuro-computing model) for software defect prediction. The overall process including data preparation, resampling, min-max normalization followed by the proposed HNC model for software defect prediction are discussed in this chapter.

3.2 Heuristic Driven Neuro-Computing Model for Software Defect Prediction

This section discusses the overall processes involved in SDP tasks. The key processes applied like data acquisition and processing, feature election, resampling and normalization are discussed, which is followed by the proposed HNC algorithm's discussion.

3.2.1 Data Acquisition and Pre-processing

In order to examine efficacy of the proposed software defect prediction model, the standard benchmark datasets been gathered from NASA PROMISE archive. There were four in all datasets were collected from the aforesaid source. These data are:

1. *Ant1.7*
2. *Camel1.6*
3. *IVY, and*
4. *JEdit.*

Noticeably, these datasets were obtained from the different software components by processing state-of-art art mining methods like Chidamber and Kamerer Java Virtual Machine (CKJM). Moreover, being developed using OOP-concept, CKJM tool extracted a total of 22 software metrics. However, these software metrics have the different level of significance towards software defect prediction. Considering this fact, in this study different feature engineering concepts including univariate logistic regression driven feature selection, resampling and min-max normalization were

performed. A snippet of these feature engineering processes is given in the subsequent sections.

3.2.2 Univariate Logistic Regression based Feature Selection

In general, Univariate logistic regression (ULR) method performs statistical analysis by employing dependent (here, per-class software reusability) as well as independent (software metrics) variables. Being a two-class problem; Normal or Defect or Fault class, the dependent variable serves two labels or values 1 or zero, signifies Normal or Defect or Fault class, respectively. Thus, it acquired the degree of meaning of each OOP CK metrics for software re-usability prediction using. Mathematically, we use (1) to estimate logistic regression value.

$$\pi(x) = \frac{e^{\alpha_0+\alpha_1 X}}{1 + e^{\alpha_0+\alpha_1 X}} \tag{3.1}$$

In (3.1), logit[π(x)] the dependent variable is stated, whereas the independent variable is represented by X.. The parameter π signifies the likelihood factor signifying the importance of each metrics. Mathematically we estimate $\pi(x)$ as per (3.2).

$$logit[\pi(x)] = \alpha_0 + \alpha_1 X \tag{3.2}$$

Let, X be the data possessing N rows and $M + 1$ columns, where M be the number of independent variables for each raw (here, 17 software metrics) presents software metrics, and β be a column vector of length $K + 1$. Additionally, there is single parameter pertaining to each M columns of the independent variable. Thus, applying logistic regression function also called $Logit$ function we obtained the log-odds of the likelihood of success to the linear component. Mathematically,

$$Logit\left(\frac{\theta_i}{1-\theta_i}\right) = \sum_{m=0}^{M} x_{im}\beta_m \qquad i = 1,2,\dots,N \tag{3.3}$$

In (3), $\left(\frac{\theta_i}{1-\theta_i}\right)$ refers a factor often stated as odds-of-an-event. Now, let y takes a value 1 for Normal or Defect or Fault class, y can be stated to have a Bernoulli distribution

with a probability parameter p. Thus, obtaining the probability parameter, also called p-value for each instance, the proposed model selects the one with $p \geq 0.05$. Though, the use of CKJM model extracts a total of 22 distinct features; however, the use of ULR feature selection method identified a total of six different OOP metrics which are significant towards defect prediction. These identified metrics were:

These software metrics are:

- *WMC- Weighted Methods per Class*
- *DIT- Depth of Inheritance Tree*
- *NOC- Number of Children*
- *CBO – Coupling between Object Classes*
- *RFC- Response for a Class*
- *LCOM-Lack of Cohesion in Methods*
- *Ca- Afferent Couplings*
- *Ce- Efferent Couplings*
- *NPM- Number of Public Methods*
- *DAM- Data Access Metric*
- *MOA- Measure of Aggregation*
- *MFA- Measure of Functional Abstraction*
- *CAM- Cohesion Among Methods of Class*
- *CC-Cyclomatic Complexity*
- *LOC- Lines of Code IC- Inheritance Coupling*
- *CBM- Coupling Between Methods*
- *AMC- Average Method Complexity*

Thus, there are 17 different characteristics in all were used for further computing.

3.2.3 Feature Re-sampling

Realizing the fact that in considered data sets (i.e., IVY, JEdit, Camel and Ant), the number of defect classes are relatively very small in comparison to the normal classes. Therefore, it signifies the probability of class-imbalance. Such class-imbalance problem can force a machine learning model to exhibit skewed performance with high false-positive performance output. Considering this fact, in this work, resampling was performed over each input data. More specifically, this work applied synthetic minority over-sampling method was applied to increase minority samples sufficiently large so as to ensure optimality of the training process. Sub-sampling can alleviate the problem of class imbalance which can't stop machine learning models to perform efficient learning (say, good learning model) and give higher accuracy. To perform up-sampling and down-sampling the proposed model considered the confidence level of 95%. Realizing the fact that merely performing over-sampling of the minority class or under-sampling of the majority class or even random resampling can't alleviate class-imbalance issues completely, as eventually turns into bias of the model towards the majority class. In such conditions, when new sample comes into the learning model, it is finally predicted as majority class due to bias towards the majority class. Such limitations can force most of the machine learning to yield false prediction output(s). Realizing these facts, in this work synthetic minority over-sampling technique was applied. To achieve it, the proposed model generated synthetic positive samples using K-nearest neighbour (k-NN) algorithm, where it employed 5-Nearest Neighbourhood to the minority "Defect or Fault" class, which was followed by equalization of the samples in such manner that it yields the number of majority class same as the number of minority class.

3.2.4 Min-Max Normalization

This is the obvious reality that in significant classification or prediction systems, particularly in enormous features-based models data imbalance and convergence are the major issues that are impeding the system's overall performance. Post feature extraction and selection the retrieved data because the pieces are of varying sizes and ranges, computing over such unstructured data might cause pre-mature convergence

and even over-fitting of the learning model. It can affect overall computational efficiency (i.e., accuracy and reliability) and therefore to alleviate it, in the proposed model the input data were processed for Min-Max normalization. A suggested Min-Max normalization algorithm, as indicated in (3.4) feature values in the range of 0 to 1 that have been mapped or normalized. The input characteristics were linearly converted and mapped using this approach in the range [0, 1]. Functionally, each data element x_i of the selected features X was mapped to the corresponding normalized value x_i' in the range of [0, 1]. The proposed model applied equation (3.4) to estimate normalized value(s) of the input data x_i.

$$Norm(x_i) = x_i' = \frac{x_i - min(X)}{max(X) - min(X)} \tag{3.4}$$

In (3.4), the information components min (X) and max(X) express the base and most extreme upsides of X, separately.

Data normalization is used in the suggested model was applied to all input benchmark datasets that resulted the following normalized outputs.

$$[D_i] = Norm(i \subset JEdit, IVY, ANT, CAMEL) \tag{3.5}$$

3.2.5 Heuristic Driven Neuro-Computing Model for SDP

Once the input data or the selected features per dataset are normalized, the proposed model projected each input data to the proposed heuristic driven neuro-computing model (HNC) to perform two-class classification. The following is a full overview of the suggested HNN model and its derivation. given as follows:

3.2.5.1 Independent and Dependent Variable Definition

It is clear that the primary goal of this study is to establish the correlation between various metrics and fault propensity. It's worth noting that the link between measurements and fault proneness at the class level is not linear. In order to overcome these challenges, a defect was used as a dependent variable in this study, whereas the specific CK metric was considered as an independent variable. Here, it is planned to foster a capacity between issue of a class and CK metrics. The investigated fault has

been described in terms of the purpose of 17 CK metrics in this thesis which can be depicted as follows:

$$Faults = f(WMC, DIT, NOC, CBO, RFC, LCOM, Ca, Ce, NPM, DAM, MOA, MFA, CAM, CC, LOC, CBM, AMC,) \quad (3.6)$$

3.2.5.2 Neu-Computing Model: Definition

A number of advances have been made towards artificial intelligence systems, basically motivated by biological neural networks and its functioning. ANN has been used by researchers from varied fields to solve a number of computational problems. The traditional mechanisms proposed to solve the computational problems have confined performance. In order to provide optimal solution ANNs has emerged as a potential technique and it has established itself as a more preferred alternative to solve major computational and decision-oriented issues. In this thesis, an improved version of ANN has served as a foundation in machine learning technology in order to accomplish software defect prediction and classification. A brief of the introduction of ANN is given in the following sections.

In fact, whatever position ANN has these days, have come across three phases. The first phase can be stated to be in the 1940s which was contributed primarily during the period of McCulloch and Pitts' [116]. Second phase could be stated to be existing in the 1960s when researchers like Rosenblatt [117] who proposed theories like perceptron convergence [117] and Minsky et al., [118] who depicted the key shortcomings of simple perceptron based NN [119]. In fact, this phase motivated majority of researchers to come out with certain optimal ANN network to contribute towards efficient computer science applications. This period lasted almost 20 years, which was then followed by third phase that was during 1980s when ANNs accomplished significant outcomes. Later, the predominant developments took place when Hopfield's energy approach was introduced [120]. In the same period a revolution came into existence with the emergence of the back-propagation algorithm which was especially employed for multilayer perceptron [121]. This approach was

later processed and optimized many time [122]. A number of researchers have provided ample of development and optimization phases of ANN [123][124].

3.2.5.3 Computational Models of Neurons

This formula neuron computes a weighted sum of its n input signals, x, j = 1, 2,…,n, as well as produces an output of 1 if the total exceeds a specified limit U, the result is 0..

$$y = \theta\left(\sum_{j=1}^{n} w_j\, x_j - u\right) \tag{3.7}$$

where θ() is a periodic function with a unit step at 0, and w_j is there a relationship between synapse weight and *j*th input. For straightforwardness of description, overall, the edge U is considered as another weight $w_0 = -U$ attached to the neuron with a constant in put $x_0 = 1$. Excitatory synapses are represented by positive weights, whereas inhibitory synapses are represented by negative weights. McCulloch and Pitts demonstrated that, in theory, suitably chosen weights let a synchronous arrangement of such neurons perform universal computations.

There's a rudimentary biological comparison here neuron: Axons and dendrites are represented by wires and linkages, synapses are represented by connection weights, and the exercise in a soma is approximated by the edge function. The McCulloch and Pitts model, However, there are a lot of basic assumptions in it that do not match actual neuron behaviour. The McCulloch-Pitts neuron has been broadened in a variety of ways. An undeniable one is to utilize actuation function other than the limit function, for example, piecewise linear, sigmoid, or Gaussian. The sigmoid function is by a wide margin the most regularly utilized in ANNs. a rigorously expanding function that displays smoothness and has the ideal asymptotic properties. The standard sigmoid function is the logistic function, as stated by

$$g(x) = \frac{1}{(1 + exp^{-\beta x})} \tag{3.8}$$

3.2.5.4 ANN Architectures

ANNs may be thought as a weighted directed graphs in which artificial neurons are nodes and directed edges are connections between neuron outputs and neuron inputs. Based on the connection pattern (architecture), ANNs can be grouped into two categories:

- *Feed-forward networks, in which graphs have no loops and*
- *Recurrent (or feedback) networks, in which loops occur because of feedback connections.*

In the most well-known group of feed-forward networks, called multi-facet perceptron, neurons are organized into layers that have unidirectional connections between them. Different connectivity yields different network behaviours. Generally speaking, feed-forward networks are static, that is, they produce only one set of output values rather than a sequence of values from a given input. Feed forward networks are memory-less in the sense that their response to an input is independent of the previous network state. Recurrent, or feedback, networks, on the other hand, are dynamic systems. When a new input pattern is presented, the neuron outputs are computed. Because of the feedback paths, the inputs to each neuron are then modified, which leads the network to enter a new state. Different network architectures require appropriate learning algorithms. The next section provides an overview of learning processes.

3.2.5.5 ANN Learning

The ability to learn is a fundamental trait of intelligence. Although a precise definition of learning is difficult to formulate, a learning process in the ANN context can be viewed as the problem of updating network architecture and connection weights so that a network can efficiently perform a specific task. The network usually must learn the connection weights from available training patterns. Performance is improved over time by iteratively updating the weights in the network. ANNs' ability to automatically learn from examples makes them attractive and exciting. Instead of following a set of rules specified by human experts, ANNs appear to learn underlying rules from the

given collection of representative examples. This is one of the major advantages of neural networks over traditional expert systems.

There are three main learning paradigms: supervised, unsupervised, and hybrid. In supervised learning, or learning with a "teacher," the network is provided with a correct answer (output) for every input pattern. Weights are determined to allow the network to produce answers as close as possible to the known correct answers. Reinforcement learning is a variant of supervised learning in which the network is provided with only a critique on the correctness of network outputs, not the correct answers themselves. In contrast, unsupervised learning, or learning without a teacher, does not require a correct answer associated with each input pattern in the training data set. It explores the underlying structure in the data, or correlations between patterns in the data, and organizes patterns into categories from these correlations. Hybrid learning combines supervised and unsupervised learning. Part of the weights is usually determined through supervised learning, while the others are obtained through unsupervised learning. Learning theory must address three fundamental and practical issues associated with learning from samples: capacity, sample complexity, and computational complexity. Capacity concerns how many patterns can be stored, and what functions and decision boundaries a network can form.

Sample complexity determines the number of training patterns needed to train the network to guarantee a valid generalization. Too few patterns may cause "over-fitting" (wherein the network performs well on the training data set, but poorly on independent test patterns drawn from the same distribution as the training patterns. Computational complexity refers to the time required for a learning algorithm to estimate a solution from training patterns. Many existing learning algorithms have high computational complexity. Designing efficient algorithms for neural network learning is a very active research topic.

There are four basic types of learning rules: error correction, Boltzmann, Hebbian, and competitive learning. In this thesis, the error correction model has been considered for learning towards SDP. A brief of error correction rules is given as follows:

3.2.5.6 Error-Correction Rules Based Learning

In the supervised learning paradigm, the network is given a desired output for each input pattern. During the learning process, the actual output y generated by the network may not equal the desired output d. The basic principle of error-correction learning rules is to use the error signal $(d - y)$ to modify the connection weights to gradually reduce this error. The perceptron learning rule is based on this error-correction principle. A perceptron consists of a single neuron with adjustable weights, $w_j, j = 1,2, \ldots, n$, and threshold U. Given an input vector$x = (x_1, x_2, \ldots, x_n)$, the net input to the neuron is

$$v = \sum_{j=1}^{n} w_j x_j - u \tag{3.9}$$

The output y of the perceptron is $+1$ if $v > 0$, and 0 otherwise. In a two-class classification problem, the perceptron assigns an input pattern to one class if$y = 1$, and to the other class if y=O. The linear equation defines the decision boundary that halves the space.

$$\sum_{j=1}^{n} w_j x_j - u = 0 \tag{3.10}$$

Rosenblatt [84] developed a learning procedure to determine the weights and threshold in a perceptron, given a set of training patterns. Note that learning occurs only when the perceptron makes an error. Rosenblatt proved that when training patterns are drawn from two linearly separable classes, the perceptron learning procedure converges after a finite number of iterations. This is the perceptron convergence theorem. In practice, it is not known whether the patterns are linearly separable. Many variations of this learning algorithm have been proposed in the literature [85] . Other activation functions that lead to different learning characteristics can also be used. However, a single-layer perceptron can only separate linearly separable patterns as long as a monotonic activation function is used. The proposed ANN model in this thesis employs a linear activation function.

Weight estimation plays a vital role in ANN learning. A number of researches have been done for optimizing weight estimation for efficient ANN learning. In this thesis, it is intended to perform software defect prediction using evolutionary computing (for weight estimation) based LM-ANN.

Before discussing the implementation of LM-ANN for defect prediction, it is important to discuss emergence of the weight estimation approach, its robustness over other existing weight estimation algorithms such as gauss Newton, gradient descent etc. In fact, LM-ANN encompasses strengths of both these algorithms and according to till available literatures, LM-ANN algorithm outperforms another existing ANN algorithm. In this thesis, it is intended to compare the performance of proposed Evolutionary Computing Based ANN algorithm with the best possible existing ANN scheme, and for this LM-ANN can be a potential competitor. The following sections discuss the different weight estimation techniques and their respective strengths as well as weaknesses. Considering limitations as well as strengths of existing approaches, how LM-ANN is derived, has been discussed in these sections.

3.2.5.7 LM-ANN Neuro-Computing for SDP Learning

LM-ANN, which is hypothesized as the best neuro-computing model so far was independently developed by Kenneth Levenberg and Donald Marquardt, provides a numerical solution to the problem of minimizing a nonlinear function. It is fast and has stable convergence. In the artificial neural-networks field, this algorithm is suitable for training small- and medium-sized problems. A number of other methods have already been developed for neural-networks training. The steepest descent algorithm, also known as the error back propagation (EBP) algorithm, dispersed the dark clouds on the field of artificial neural networks and could be regarded as one of the most significant breakthroughs for training neural networks. Many improvements have been made to EBP, but these improvements are relatively minor. The EBP algorithm is still widely used today; however, it is also known as an inefficient algorithm because of its slow convergence. There are two main reasons for the slow convergence: the first reason is that its step sizes should be adequate to the gradients (Figure 3.2). Logically, small step sizes should be taken where the gradient is steep so

as not to rattle out of the required minima. So, if the step size is a constant, it needs to be chosen small. Then, in the place where the gradient is gentle, the training process would be very slow. The second reason is that the curvature of the error surface may not be the same in all directions, thus creating "error valley" problem that might result into the slow convergence.

The slow convergence of the steepest descent method can be greatly improved by the Gauss–Newton algorithm. Using second-order derivatives of error function to "naturally" evaluate the curvature of error surface, The Gauss–Newton algorithm can find proper step sizes for each direction and converge very fast; especially, if the error function has a quadratic surface, it can converge directly in the first iteration. But this improvement only happens when the quadratic approximation of error function is reasonable. Otherwise, the Gauss–Newton algorithm would be mostly divergent.

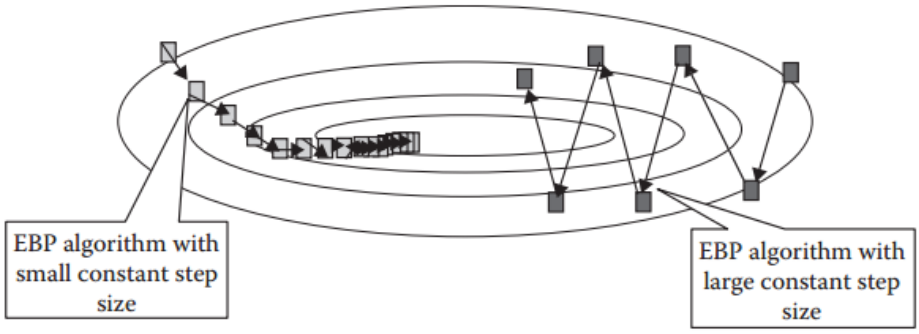

Figure 3.1 Searching process of the steepest descent method

3.2.5.8 Modelling LM-ANN Neuro-Computing Model

The LM algorithm blends the steepest descent method and the Gauss–Newton algorithm. Fortunately, it inherits the speed advantage of the Gauss–Newton algorithm and the stability of the steepest descent method. It's more robust than the Gauss–Newton algorithm, because in many cases it can converge well even if the error surface is much more complex than the quadratic situation. Although the LM algorithm tends to be a bit slower than Gauss–Newton algorithm, it converges much faster than the steepest descent method. The basic idea of the LM algorithm is that it performs a combined training process: around the area with complex curvature, the LM algorithm

switches to the steepest descent algorithm, until the local curvature is proper to make a quadratic approximation; then it approximately becomes the Gauss–Newton algorithm, which can speed up the convergence significantly. In this section of the presented thesis, the derivation of the Levenberg–Marquardt algorithm has been discussed in four parts:

1. *Steepest descent algorithm,*
2. *Newton's method,*
3. *Gauss–Newton's algorithm, and*
4. *Levenberg– Marquardt algorithm.*

Before the derivation, let us introduce some commonly used indices. Here, p is the index of patterns, from 1 to P, where P is the number of patterns, m is the index of outputs, from 1 to M, where M is the number of outputs. The other variables i and j are the indices of weights, from 1 to N, where N is the number of weights. k is the index of iterations.

Other indices will be explained in related places. Sum square error (SSE) is defined to evaluate the training process. For all training patterns and network outputs, it is calculated by following equation:

$$E(x,w) = \frac{1}{2}\sum_{p=1}^{P}\sum_{m=1}^{M} e_{p,m}^2 \tag{3.11}$$

where ,x is the input vector, w is the weight vector, $e_{p,m}$ is the training error at output m which after applying pattern p turns out to be:

$$e_{p,m} = d_{p,m} - 0_{p,m} \tag{3.12}$$

where, d is the desired output vector and o is the actual output vector

A. Steepest Descent Algorithm

The steepest descent algorithm is a first-order algorithm that uses the first-order derivative of total error function to find the minima in error space. Normally, gradient g is defined as the first-order derivative of total error function (3.13):

$$g = \frac{\partial E(x,w)}{\delta w} = \left[\frac{\partial E}{\partial w_1} \quad \frac{\partial E}{\partial w_2} \cdots \frac{\partial E}{\partial w_N}\right]^T \tag{3.13}$$

With the definition of gradient g in (3.13), the update rule of the steepest descent algorithm could be written as

$$w_{k+1} = w_k - \alpha g_k \tag{3.14}$$

where α is the learning constant (step size).

The training process of the steepest descent algorithm is asymptotic convergence. Around the solution, all the elements of gradient vector would be very small and there would be a very tiny weight change.

B. Newton's Method

Newton's method assumes that all the gradient components $g_1, g_2, \ldots, g_N$ are functions of weights and all weights are linearly independent:

$$\begin{cases} g_1 = F_1(w_1, w_2 \cdots w_N) \\ g_2 = F_2(w_1, w_2 \cdots w_N) \\ \cdots \\ g_N = F_N(w_1, w_2 \cdots w_N) \end{cases} \tag{3.15}$$

where $F_1, F_2, \ldots, F_N$ are nonlinear relationships between weights and related gradient components. Unfold each g_i $(i = 1, 2, \ldots, N)$ in Equations (3.15) by Taylor series and take the first-order approximation:

$$\begin{cases} g_1 \approx g_{1,0} + \frac{\partial g_1}{\partial w_1}\Delta w_1 + \frac{\partial g_1}{\partial w_2}\Delta w_2 + \cdots + \frac{\partial g_1}{\partial w_N}\Delta w_N \\ g_2 \approx g_{2,0} + \frac{\partial g_2}{\partial w_1}\Delta w_1 + \frac{\partial g_2}{\partial w_2}\Delta w_2 + \cdots + \frac{\partial g_2}{\partial w_N}\Delta w_N \\ \cdots \\ g_N \approx g_{N,0} + \frac{\partial g_N}{\partial w_1}\Delta w_1 + \frac{\partial g_N}{\partial w_2}\Delta w_2 + \cdots + \frac{\partial g_N}{\partial w_N}\Delta w_N \end{cases} \tag{3.16}$$

By combining the definition of gradient vector g in (3.13), it could be determined that

$$\frac{\partial g_i}{\partial w_j} = \frac{\partial\left(\frac{\partial E}{\partial w_j}\right)}{\partial w_j} = \frac{\partial^2 E}{\partial w_i \partial w_j} \tag{3.17}$$

Now, by inserting Equation (3.17- 3.20):

$$\begin{cases} g_1 \approx g_{1,0} + \dfrac{\partial^2 E}{\partial w_1^2}\Delta w_1 + \dfrac{\partial^2 E}{\partial w_1 \partial w_2}\Delta w_2 + \cdots + \dfrac{\partial^2 E}{\partial w_1 \partial w_N}\Delta w_N \\ g_2 =\approx g_{2,0} + \dfrac{\partial^2 E}{\partial w_2 \partial w_1}\Delta w_1 + \dfrac{\partial^2 E}{\partial w_2^2}\Delta w_2 + \cdots + \dfrac{\partial^2 E}{\partial w_2 \partial w_N}\Delta w_N \\ \cdots \\ g_N =\approx g_{N,0} + \dfrac{\partial^2 E}{\partial w_N \partial w_1}\Delta w_1 + \dfrac{\partial^2 E}{\partial w_N \partial w_2}\Delta w_2 + \cdots + \dfrac{\partial^2 E}{\partial w_N^2}\Delta w_N \end{cases} \tag{3.18}$$

Comparing with the steepest descent method, the second-order derivatives of the total error function need to be calculated for each component of gradient vector. In order to get the minima of total error function E, each element of the gradient vector should be zero. Therefore, left sides of the equations (3.18) are all zero, then

$$\begin{cases} 0 \approx g_{1,0} + \dfrac{\partial^2 E}{\partial w_1^2}\Delta w_1 + \dfrac{\partial^2 E}{\partial w_1 \partial w_2}\Delta w_2 + \cdots + \dfrac{\partial^2 E}{\partial w_1 \partial w_N}\Delta w_N \\ 0 \approx g_{2,0} + \dfrac{\partial^2 E}{\partial w_2 \partial w_1}\Delta w_1 + \dfrac{\partial^2 E}{\partial w_2^2}\Delta w_2 + \cdots + \dfrac{\partial^2 E}{\partial w_2 \partial w_N}\Delta w_N \\ \cdots \\ 0 \approx g_{N,0} + \dfrac{\partial^2 E}{\partial w_N \partial w_1}\Delta w_1 + \dfrac{\partial^2 E}{\partial w_N \partial w_2}\Delta w_2 + \cdots + \dfrac{\partial^2 E}{\partial w_N^2}\Delta w_N \end{cases} \tag{3.19}$$

By combining Equation 3.20 with 3.19,

$$\begin{cases} -\dfrac{\partial E}{\partial w_1} = -g_{1,0} \approx \dfrac{\partial^2 E}{\partial w_1^2}\Delta w_1 + \dfrac{\partial^2 E}{\partial w_1 \partial w_2}\Delta w_2 + \cdots + \dfrac{\partial^2 E}{\partial w_1 \partial w_N}\Delta w_N \\ -\dfrac{\partial E}{\partial w_2} = -g_{2,0} \approx \dfrac{\partial^2 E}{\partial w_2 \partial w_1}\Delta w_1 + \dfrac{\partial^2 E}{\partial w_2^2}\Delta w_2 + \cdots + \dfrac{\partial^2 E}{\partial w_2 \partial w_N}\Delta w_N \\ \cdots \\ -\dfrac{\partial E}{\partial w_N} = -g_{N,0} \approx \dfrac{\partial^2 E}{\partial w_N \partial w_1}\Delta w_1 + \dfrac{\partial^2 E}{\partial w_N \partial w_2}\Delta w_2 + \cdots + \dfrac{\partial^2 E}{\partial w_N^2}\Delta w_N \end{cases} \tag{3.20}$$

There are N equations for N parameters so that all Δw_i can be calculated. With the solutions, the weight space can be updated iteratively.

Equations 3.20 can be also written in matrix form (3.21).

$$\begin{bmatrix} -g_1 \\ -g_2 \\ \cdots \\ -g_N \end{bmatrix} = \begin{bmatrix} -\dfrac{\partial E}{\partial w_1} \\ -\dfrac{\partial E}{\partial w_2} \\ \cdots \\ -\dfrac{\partial E}{\partial w_N} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial^2 E}{\partial w_1^2} & \dfrac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \dfrac{\partial^2 E}{\partial w_1 \partial w_N} \\ \dfrac{\partial^2 E}{\partial w_2 \partial w_1} & \dfrac{\partial^2 E}{\partial w_2^2} & \cdots & \dfrac{\partial^2 E}{\partial w_2 \partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \dfrac{\partial^2 E}{\partial w_N \partial w_1} & \dfrac{\partial^2 E}{\partial w_N \partial w_2} & \cdots & \dfrac{\partial^2 E}{\partial w_N^2} \end{bmatrix} \times \begin{bmatrix} \Delta w_1 \\ \Delta w_2 \\ \cdots \\ \Delta w_N \end{bmatrix} \tag{3.21}$$

where the square matrix is Hessian matrix:

$$H = \begin{bmatrix} \dfrac{\partial^2 E}{\partial w_1^2} & \dfrac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \dfrac{\partial^2 E}{\partial w_1 \partial w_N} \\ \dfrac{\partial^2 E}{\partial w_2 \partial w_1} & \dfrac{\partial^2 E}{\partial w_2^2} & \cdots & \dfrac{\partial^2 E}{\partial w_2 \partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \dfrac{\partial^2 E}{\partial w_N \partial w_1} & \dfrac{\partial^2 E}{\partial w_N \partial w_2} & \cdots & \dfrac{\partial^2 E}{\partial w_N^2} \end{bmatrix} \tag{3.22}$$

Now, by combining Equations 3.13 and 3.22 with equation (3.21),

$$-g = H\Delta w \tag{3.23}$$

So

$$\Delta w = -H^{-1} g \tag{3.24}$$

Therefore, the update rule for Newton's method is

$$w_{k+1} = w_k - H_k^{-1} g_k \tag{3.24}$$

As the second-order derivatives of total error function, Hessian matrix H gives the proper evaluation on the change of gradient vector. By comparing Equations (3.14 and 3.24), one may notice that well matched step sizes are given by the inverted Hessian matrix.

C. Gauss–Newton Algorithm

If Newton's method is applied for weight updating, in order to get Hessian matrix$H$, the second-order derivatives of total error function have to be calculated and it could be very complicated. In order to simplify the calculating process, Jacobian matrix J is introduced as follows:

$$J = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w_1} & \frac{\partial e_{1,1}}{\partial w_2} & \cdots & \frac{\partial e_{1,1}}{\partial w_N} \\ \frac{\partial e_{1,2}}{\partial w_1} & \frac{\partial e_{1,2}}{\partial w_2} & \cdots & \frac{\partial e_{1,2}}{\partial w_N} \\ \cdots & \cdots & & \cdots \\ \frac{\partial e_{1,M}}{\partial w_2} & \frac{\partial e_{1,M}}{\partial w_2} & \cdots & \frac{\partial e_{1,M}}{\partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{p,1}}{\partial w_1} & \frac{\partial e_{p,1}}{\partial w_2} & \cdots & \frac{\partial e_{p,1}}{\partial w_N} \\ \frac{\partial e_{P,2}}{\partial w_1} & \frac{\partial e_{P,2}}{\partial w_2} & \cdots & \frac{\partial e_{p,2}}{\partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{P,M}}{\partial w_1} & \frac{\partial e_{P,M}}{\partial w_2} & & \frac{\partial e_{P,M}}{\partial w_N} \end{bmatrix} \tag{3.25}$$

Performing integration of equations (3.11) and (3.12), the gradient vector's elements can be obtained as follows:

$$g_i = \frac{\partial E}{\partial w_i} = \frac{\partial \left(\frac{1}{2}\sum_{p=1}^{p}\sum_{m=1}^{M} e_{p,m}^2\right)}{\partial w_i} = \sum_{p=1}^{P}\sum_{m=1}^{M}\left(\frac{\partial e_{p,m}}{\partial w_i} e_{P,1}\right. \tag{3.26}$$

Now, amalgamating (3.25) and (3.26), the relationship between Jacobian matrix J and gradient vector g can be obtained as follows:

$$g\,J \;=\; e \tag{3.27}$$

where the error vector e can have the following presentation:

$$e = \begin{bmatrix} e_{1,1} \\ e_{1,2} \\ \cdots \\ e_{1,M} \\ \cdots \\ e_{P,1} \\ e_{P,2} \\ \cdots \\ e_{P,M} \end{bmatrix} \tag{3.28}$$

Now, using (3.21), the comprising elements of Hessian matrix can be estimated as follows

$$h_{i,j} = \frac{\partial^2 E}{\partial w_i \partial w_j} = \frac{\partial^2 \left(\frac{1}{2}\sum_{p=1}^{p}\sum_{m=1}^{M} e_{p,m}^2\right)}{\partial w_i \partial w_j} = \sum_{p=1}^{P}\sum_{m=1}^{M} \frac{\partial e_{p,m}\partial e_{p,m}}{\partial w_i \partial w_j} + S_{i,j} \tag{3.29}$$

where $S_{i,j}$ can be obtained by

$$S_{i,j} = \sum_{p=1}^{P}\sum_{m=1}^{M} \frac{\partial^2 e_{p,m}}{\partial w_i \partial w_j} e_{p,m} \tag{3.30}$$

In general, $S_{i,j}$ for Newton approach is used to be near zero, and thus the relationship between Hessian matrix H and Jacobian matrix J can be obtained as follows:

$$H \approx J^T J \tag{3.31}$$

Combining (3.24), (3.26), and (3.31), weight update rule for Gauss Newton can be retrieved as

$$w_{k+1} = w_k - (J_k^T J_k)^{-1} J_k e_k \tag{3.32}$$

This is the matter of fact that the advantage of the Gauss–Newton algorithm over the standard Newton's method (3.24) is that the former does not require the calculation of second-order derivatives of the total error function, by introducing Jacobian matrix J instead. However, the Gauss–Newton algorithm still faces the same convergent problem like the Newton algorithm for complex error space optimization. Mathematically, the problem can be interpreted as the matrix $\boldsymbol{J^T J}$ may not be invertible.

D. Levenberg Marquardt Algorithm

In order to make sure that the approximated Hessian matrix $\boldsymbol{J^T J}$ is invertible, Levenberg–Marquardt algorithm introduces another approximation to Hessian matrix:

$$H \approx J^T J + \mu I \tag{3.33}$$

where μ is always positive, called combination coefficient I is the identity matrix

Taking into consideration of the equation (3.33), one may notice that the elements on the main diagonal of the approximated Hessian matrix will be larger than zero. Therefore, with this approximation (Equation 3.33), it can be sure that matrix H is always invertible.

Now, combining Equations 3.32 and 3.33, the update rule of Levenberg–Marquardt algorithm is presented as:

$$w_{k+1} = w_k - (J_k^T J_k + \mu I)^{-1} J_k e_k \tag{3.34}$$

As the combination of the steepest descent algorithm and the Gauss–Newton algorithm, the Levenberg-Marquardt algorithm switches between the two algorithms during the training process. When the combination coefficient μ is very small (nearly zero), Equation 3.28 is approaching to Equation 3.26 and Gauss–Newton algorithm is used. When combination coefficient μ is very large, Equation 3.28 approximates to Equation 3.8 and the steepest descent method is used. If the combination coefficient μ in Equation 3.34 is very big, it can be interpreted as the learning coefficient in the steepest descent method (3.34):

$$\alpha = \frac{1}{\mu} \tag{3.35}$$

In this thesis, in order to implement the Levenberg–Marquardt algorithm for ANN training to be used for software defect prediction and classification, two problems have been solved. These are the effective calculation of the Jacobian matrix, and organizing the training process iteratively for weight updating.

3.2.5.9 Heuristic Driven LM-ANN Neuro-Computing for Software Defect Prediction

Functionally, ANN mimics human brain functionalities to learn over certain input data or patterns. Thus, learning over such input patterns it classifies unknown input into target categories. An illustration of ANN model is given in Figure 3.2. As depicted in the figure 3.2, ANN comprise three layers; input layer, hidden layer and output layer. Considering architecture of ANN, it embodies multiple neurons representing the input data to be further processed at the distinct intermediate layers (say, hidden layers) for

classification (at the output layer). To learn over the input data, ANN applies error-reduction method, where during learning it estimates the difference between the expected output and the observed output (signifying error). The learning process continues till the error output becomes zero or near zero. Thus, achieving zero-error the outputs at the output layer is predicted as the final output. In this work, the ANN algorithms perform two-class classification, where it classifies each class as Normal Class or Faulty Class and label them as "1" and "0", respectively. Being a two-class classification problem, the ANN designs have the one output layer (Figure 3.2). In the proposed neuro-computing or allied learning model, the final features pertaining to each class of the software are fed as input to the ANN (Figure 3.2), while the number of hidden layers is varied. At the input layer of the ANN, it applied linear activation function, which generates output same as the input (i.e., $O_o = I_i$), while the output of the hidden layer is fed to the input of the output layer. Noticeably, output layer of the ANN applies Sigmoid function (20) to generate O_h.

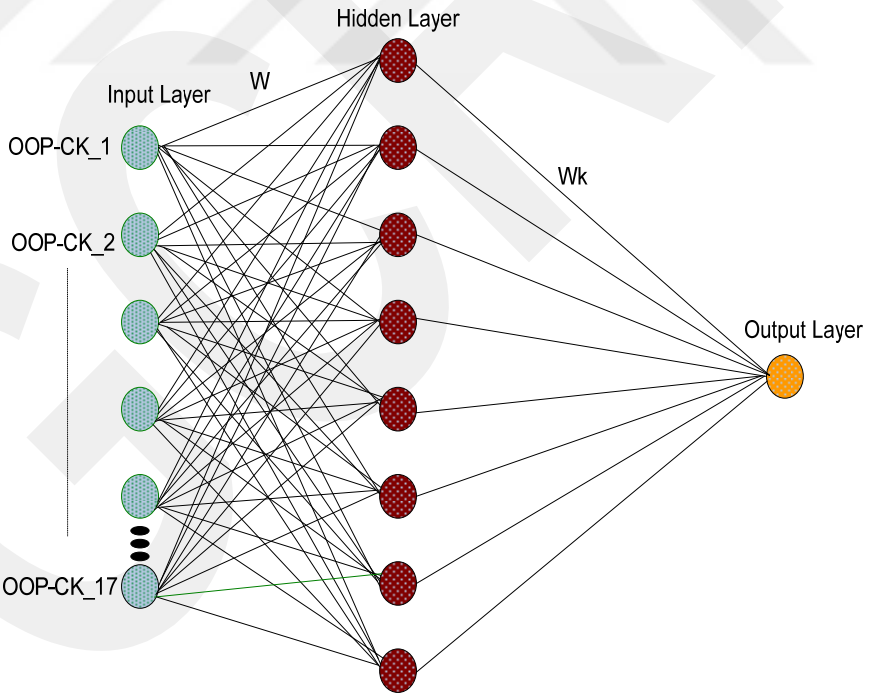

Figure 3.2 An illustration of ANN architecture with single hidden layer with one output node

$$O_h = \frac{1}{1 + e^{-I_h}} \tag{3.36}$$

In (3.36), I_h represents the input at the hidden layer. ANN is often defined as $Y' = f(W, X)$ where Y′ states the output vector, while X and W presents the allied input and the weight values, respectively. Functionally, ANN applies certain error function such as mean square error (MSE) to achieve the higher accuracy, which is estimated using (3.37).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i^{'} - y_i\right)^2 \tag{3.3 7}$$

In (3.37), y is the observed value, while $y_i^{'}$ is the expected value.

In reference to the neuro-computing architecture used in this work (Figure 3.2), there are 17 input nodes have been defined that takes 17 CK matrix having multiple classes as individual input. Since, in the proposed ANN model, the expected outputs are either FAULTY or NO-FAULTY, therefore only one output node is needed. Here, a total of 19 hidden layers were considered so as to avoid unwanted computational complexity. Thus, in the defined ANN architecture, 342 weights $(input\ node + Output\ Node) * hidden\ node)$ are required to be estimated for fault prediction and classification. Now, to perform learning over the targeted neuro-model (Figure 3.2), the proposed neuro-computing approach requires estimating 342 weight parameters in each iteration, which can force the model to undergo convergence and local minima, thus impacting overall performance. Considering this fact, in this research heuristic model (here, genetic algorithm) was applied to tune the aforesaid 342 weight parameters continuously so as to avoid local minima and convergence. It is hypothesized to yield better performance.

The detailed discussion of the proposed heuristic driven neuro-computing model (HNC) is discussed as follows:

Typically, the neuro-computing model defined in (3.34), often called LM-ANN algorithm performs localization of the bare minimum value of multivariate function in a repetitive manner, which is expressed as the sum of squares of non-linear real-

valued functions. Similar to GD algorithm, LM-ANN updates the weights during NN learning process. Considering the performance novelty, the proposed LM-ANN algorithm comprises the functional ability of Steepest Descent and Gauss Newton method. The proposed LM algorithm updates the weight vector using equation (3.34) $[W_{k+1} = W_k - \left(J_k^T J_k + \mu I\right)^{-1} J_k e_k]$, where W_{k+1} is the updated weights, W_k is the current weights, I represents the identity or unit matrix, J is the Jacobian matrix and μ, the combination coefficient is always positive. With μ as very small, it functions as Gauss Newton method while making μ as very large makes it functional as Gradient descent method. The Jacobian matrix derived as given as:

$$= J \begin{bmatrix} \frac{d}{dW_1}(E_{1,1}) & \frac{d}{dW_2}(E_{1,1}) & \cdots & \frac{d}{dW_N}(E_{1,1}) \\ \frac{d}{dW_1}(E_{1,2}) & \frac{d}{dW_2}(E_{1,2}) & \cdots & \frac{d}{dW_N}(E_{1,2}) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{d}{dW_1}(E_{P,M}) & \frac{d}{dW_2}(E_{P,M}) & \cdots & \frac{d}{dW_N}(E_{P,M}) \end{bmatrix} \quad (3.38)$$

In (3.38) N refers the weight counts and the input data patterns of the proposed CK metrics are P. The output patterns are indicated by M.

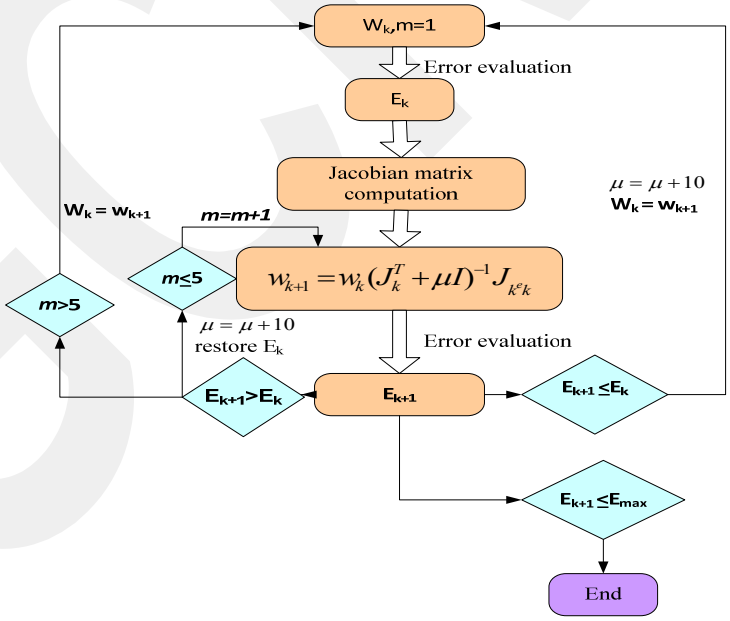

Figure 3.3 HCN Model: $\boldsymbol{W_k}$ is the current weight, $\boldsymbol{W_{k+1}}$ is the next weight, $\boldsymbol{E_{k+1}}$ is the current total error, and $\boldsymbol{E_k}$ is the final error

The overall training function by the targeted LM-ANN neuro-computing model executes the following mechanism (Figure 3.3). Noticeably, the heart of this model (Figure 3.3) can be found in equation (3.34), which is used to update the weight parameters for continuous feature learning. However, realising the fact that estimating 342 weight parameters over each iteration can force LM-ANN (Figure 3.3) to undergo local minima and convergence and therefore, this study applies a heuristic model named genetic algorithm. Functionally, in the proposed model, the applied heuristic model estimates the optimal set of weight parameters dynamically to tune the weight parameters (Equation (3.34)) that helps efficient learning to yield superior results.

3.2.5.10 Heuristic Driven LM-ANN Weight Tuning

Considering above discussed learning problems, where estimating a significantly large number of weight parameters can be difficult and hence to alleviate resulting local minima and convergence problems, this work applied a heuristic concept named Genetic algorithm. Genetic algorithm, which is an evolutionary computing concept applied Darvins principle of selection to achieve the set of optimal solution(s) from the probable sub-solution. In sync with the targeted LM-ANN optimization task, here GA intends to improve the learning of ANN with optimal set of weight parameters to achieve higher accuracy. In general, GA hypothesizes that over increasing generation, the fitness of the solution can be improved. And therefore, in the proposed method initially it deploys the random set of weights as input population. Here, each input population signifies chromosomes or a set of solutions. Here, the term solution signifies a chromosome possessing binary string's form in which all allied variables are encoded. In other words, this population or chromosome signifies an individual with encoded feature whether it is accepted or rejected based on certain defined fitness value. Once performing population generation, it estimates the fitness value of each population or candidate. Noticeably, the fitness value signifies an intended or predefined function characterising suitability of the candidate to become the solution. It performs fitness estimation for each candidate or chromosome, and the individual with superior fitness value is retained while the one with inferior fitness is dropped from further computation. Accordingly, based on the assessed fitness values, the next

generation posterity are generation by perform crossover or reproduction. To perform next generation solution creation, GA applies two key functions named crossover probability and the mutation probability. Thus, the process of suitable population process continues till one attains the optimal target value also called stopping criteria. In other words, after every generation GA estimates the suitable set of weight parameters which are examined for their efficacy (say, fitness value (here, low MSE)) and this process is continued over generation, till it provided the set of optimal weights giving error as zero or near zero for ANN-learning.

In sync with the neuro-computing model, comprising $i - h - o$ network configuration with i input layer, h hidden layer and O output layer, the proposed model feeds each of the software metrics as input to each input neuron. Since, in the proposed model, a total of 17-fetaures were given as input, it generated a neuro-computing architecture of 17-19-1 where the total number of weights required were N (3.39).

$$N = (i + O) * h \tag{3.39}$$

Thus, to implement learning enhancement where adjusting or tuning weights is must, the proposed model considered each weight value as a gene in the chromosomes. Thus, for the complete gene length l, the length of the chromosome L_{Chrom} is estimated as (3.40).

$$L_{Chrom} = N * l = (i + O) * h * l \tag{3.40}$$

Now, considering all chromosomes as input weight and population, it estimated the fitness over each generation to assess the suitability to achieve goal (low MSE). Thus, applying weight update or tuning over each generation, the proposed GA model enabled estimation of the optimal set of weight parameters which yields minimum RMSE. Noticeably, in the proposed model the weights (W_k) were updated or tuned as per (3.41).

$$W_k = \begin{cases} if\ 0 \leq\ x_{kl+1} < 5 \\ -\dfrac{x_{kl+2*}10^{l-2} + x_{kl+3*}10^{l-3} + \cdots + x_{(k+1)l}}{10^{l-2}} \\ if\ 5 <= x_{kl+l} <= 9 \\ +\dfrac{x_{kl+2*}10^{l-2} + x_{kl+3*}10^{l-3} + \cdots + x_{(k+1)l}}{10^{l-2}} \end{cases} \quad (3.41)$$

The pseudo-code applied towards the proposed heuristic driven neuro-computing (HNN) is given in Figure 3.4.

Algorithm for Fitness Estimation

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

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

Where $\bar{I}_i, \bar{T}_i$ state the input and output pairs of the $i - h - o$ configuration of neural network.

Phase1: Determine weights $\bar{W}_i$ for C_i by equation (4.3)

Phase2: Expecting $\bar{W}_i$ be a constant weight, perform training of N input as well as compute output O_i

Phase3: Determine MSE E_j for all input instance j , $E_j = (T_{ji} - O_{ji})$

Phase4: Determine RMSE of chromosome $C_i E_i = \sqrt{\frac{\sum_{j=1}^{j=N} E_j}{N}}$

Where N indicates the total number of training records (data)

Phase5: Determine the value of fitness for chromosome $C_i F_i = \frac{1}{E_i} = \frac{1}{\sqrt{\frac{\sum_{j=1}^{j=N} E_j}{N}}}$

Figure 3.4 Pseudo code for HNC

To be noted, in majority of the existing heuristic based (especially GA driven models) models, authors have applied crossover and mutation probability as 0.8, 0.2 or 0.6, 0.4 pair (say, pareto combination), respectively. However, this approaches often leads exceedingly rise in search space due to continuous accumulation of the candidate

space (in each generation). Eventually, this itself raises a problem of convergence and local minima, which can greatly impact the overall performance. Considering this fact, in this thesis an improved GA model is designed by applying dynamic crossover and mutation probability function. In this work, these parameters (i.e., probability of crossover (P_c) and mutation and mutation (P_m)) were estimated dynamically to achieve optimal solution, without undergoing any convergence or local minima condition. Mathematically, it applied equation (3.42) to perform P_c and P_m update.

$$(P_c)_{k+1} = (P_c)_k - \frac{K_1 * x}{5}$$
$$(P_m)_{k+1} = (P_m)_k - \frac{K_2 * x}{5} \quad (3.42)$$

In (3.42) $(P_c)_{k+1}$ and $(P_m)_{k+1}$ represent the revised crossover and mutation chances, respectively. The remaining variables$(P_c)_k$ and $(P_m)_k$are crossover and mutation probabilities at the present time, K_1 and K_2 it may be any positive constant and x stands for the number of chromosomes having comparative fitness value. Unlike static threshold driven optimization (say, stopping criteria as the total number of iterations or generations), the proposed HNC model continues weight parameter tuning in (equation 3.34 and Figure 3.3), till 95% of chromosomes possess the same fitness value. Thus, once it reaches the stopping criteria function, the optimized weight parameters are updated to the neuro-computing model derived in (3.34) and Figure 3.3. Subsequently, with the updated weight parameters the proposed HNC model performs classification and classifies each class as Normal or Faulty or Defect Class and labels them as 0 and 1, respectively. The outputs are generated at the output layer of the model (Figure 3.2), O_o.

The overall process of the targeted software defect prediction is depicted in the following step-wise manner.

Since, the proposed HNC model applied improved GA to perform tuning, at first it required input specifications including the initial population, crossover and mutation probability etc. Thus, inputting these values, the proposed improved GA model performed the following tasks.

Phase-1 Population Initialization:

In this work, considering the target values for optimization (here 342 weights are to be estimated), the proposed model considered a total of 350 chromosomes, which were generated randomly so as to perform further competition. The arbitrarily generated chromosomes performed crossover in sync with the updated crossover and mutation probability values as given in equation (3.42).

Phase-2 Weight Estimation and Update in LM-ANN (Equation (3.34 and Figure 3.3)):

In the proposed HNC model, to alleviate any probability of local minima and convergence caused false positive performance, the proposed model tuned weight parameters W_k in (3.34) as per Figure 3.3, to learn and perform two-class classification. Here, the updated weight parameters were applied or tuned for both input-hidden layer weights as well as hidden-output layer weight values.

Phase-3 Fitness Estimation:

Once estimating the weight values, the fitness value was obtained for each chromosome (representing each sub-solution for the targeted weight parameter) so as to minimize error (RMSE) (equation (3.37)).

Phase-4 Chromosome Update and Mutation:

Based on the fitness values for each chromosome, the proposed heuristic model updated the rank of each chromosome where only the chromosomes which higher fitness were retained, while the low fitness candidates were dropped using mutation function. Thus, the chromosomes with higher fitness replaces the chromosome with lower fitness value.

Phase-5 Crossover:

The proposed heuristic model performed two-point crossover with the retained chromosomes or candidates. To improve efficiency, $P_c$and P_m were updated

dynamically using (3.42). Note, the variable n signifies the total number of chromosomes possessing similar fitness.

Noticeably, in the proposed heuristic driven neuro-computing model, the weight update in (3.34) continues until 95% of the chromosomes possess the same fitness value. Exceeding this value might force model to undergo convergence problem.

Phase-6 Software Defect Prediction:

With the updated weight parameters, the proposed neuro-computing model performs two-class classification Thus, the proposed model classifies each class as "Normal Class" and "Faulty" class and labels each class as 0 and 1, respectively.

Phase-7 Confusion Matrix:

Once classifying each class as "Normal Class" and "Faulty" class, foe each input data, confusion matrix was obtained, which was subsequently used to perform statistical performance assessment when it comes to accuracy, precision, recall and F-Measure.

3.3 Summary

This chapter discussed the overall proposed model and its implementation to accomplish targeted software defect prediction tasks. The detailed discussion of the processes involved such as data acquisition, feature selection, resampling and normalization followed by HNC based classification were discussed in this chapter. The simulated results and allied discussions are given in the next chapter (Chapter4).

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Background

This section primarily discusses the simulation results and allied statistical inferences, signifying the relative efficacy the suggested HNC-SDP model's performance in comparison to existing models state-of-art ways. To assess relative performance analysis, four different input benchmark dataset named IVY, CAMEL, ANT and JEDIT were employed. Noticeably, these input data were obtained from NASA PROMISE repository for SDP. Overall, the proposal HNC-SDP model was developed using MATLAB2015b software tool, which was simulated with Microsoft Windows 10 Operating system armoured with 8GB RAM and Intel i5 processor. The comprehensive examination of the overall proposed model and allied statistical inferences are described in the following sections.

4.2 Characterization of Performance

To evaluate the performance of the suggested HNC-SDP model, the simulations were made for each data distinctly, and accordingly their corresponding confusion metrics were obtained. To characterise performance statistical method named confusion metrics was obtained for each test cases. Before discussing the results and allied inferences, a snippet of the confusion metrics obtained and corresponding statistical parameters (i.e., accuracy, precision, recall and F-Measure) is given in Table 4.1.

In this work, for each simulation case (say, simulation with JEDIT, Ant, Camel and IVY data, distinctly) confusion matrix measurement were obtained characterizing true positive (TP), true negative (TN), false positive (FP) and false negative (FN). After recovering these matrix values for each base learner and group classifier, the efficiency in terms of accuracy, precision, recall, and F-Measure was achieved. The meanings of these exhibition variables are given in Table 4.1.

Table 4.1 Performance Parameters

Parameter	Mathematical Expression	Definition
Accuracy	$\frac{(TN+TP)}{(TN+FN+FP+TP)}$	Indicates the percentage of predicted fault prone modules that are examined out of all modules.
Precision	$\frac{TP}{(TP+FP)}$	Indicates the extent to which repeated observations under test conditions provide the same findings.
F-measure	$2.\frac{Recall.Precision}{Recall+Precision}$	It creates a single score by combining the precision and recall numeric values, which is specified as the harmonic mean of the recall and precision.
Recall	$\frac{TP}{(TP+FN)}$	It specifies the number of objects that must be listed.

Since, this study applied multiple input data and allied performance assessment were made, the overall performance characterization is done in terms of the following:

1. *Intra-Model Comparison, and*
2. *Inter-Model Comparison.*

Here, intra-model comparison performs statistical performance assessment with the different input dataset when it comes to accuracy, precision, recall and F-Measure. On the contrary, inter-model assessment exhibits relative performance analysis with the different existing methods. The in-depth examination of simulated findings and allied significance is described as follows:

4.2.1 Intra-Model Performance Characterization

In this assessment, the performance analysis is done for the different input datasets. Here, the only motive is to evaluate adequacy of the proposed model over the various set of inputs and corresponding strengths or weakness identification, Moreover, it can help identifying the average performance by the proposed system, which can be later used for relative performance assessment (say, inter-model characterization).

4.2.1.1 Test Case-1 IVY Dataset

The confusion metrics obtained for the IVY dataset for faulty and non-faulty (i.e., normal class) is given at the Table 4.2 and Table 4.2. Noticeably, to compare and contrast results, the confusion metrics before HNC-SDP as well as post HNC-SDP execution are obtained. It can help understanding classification efficacy of the proposed model (i.e., HNC-SDP).

Table 4.2 Confusion matrix for IVY dataset before HNC-SDP execution

	Normal	**Fault/Defect**
Normal	481	0
Fault/Defect	40	0

Table 4.3 Confusion matrix for IVY data after prediction

	Normal	**Fault/Defect**
Normal	311	1
Fault/Defect	36	4

Table 4.4 represents the statistical results obtained when it comes to accuracy, precision, recall and F-Measure. Observing the results, it can be found that the accuracy achieved by the proposed HNC-SDP model is 88.35%, while precision, F-

Measure and recall values are 99.36%, 93.8% and 88.83%, respectively. Noticeably, in addition to the higher accuracy, precision and recall (say, sensitivity), the higher value of F-Measure (0.93) means power of the proposed model considerably under class-imbalanced condition. It is always hypothesized that a neuro-computing model can be stated as "Efficiently learnt", only when it exhibits decreasing RMSE or MSE values over generations. Considering this fact, MSE variation was plotted for the different datasets. As depicted in Figure 4.1, it can easily be found that the MSE decreases over increasing computation. It helps achieving superior performance.

Table 4.4 Cumulative HNC-SDP Performance evaluation for IVY data

Accuracy	Precision	F-Measure	Recall
0.8835	0.9936	0.9380	0.8883

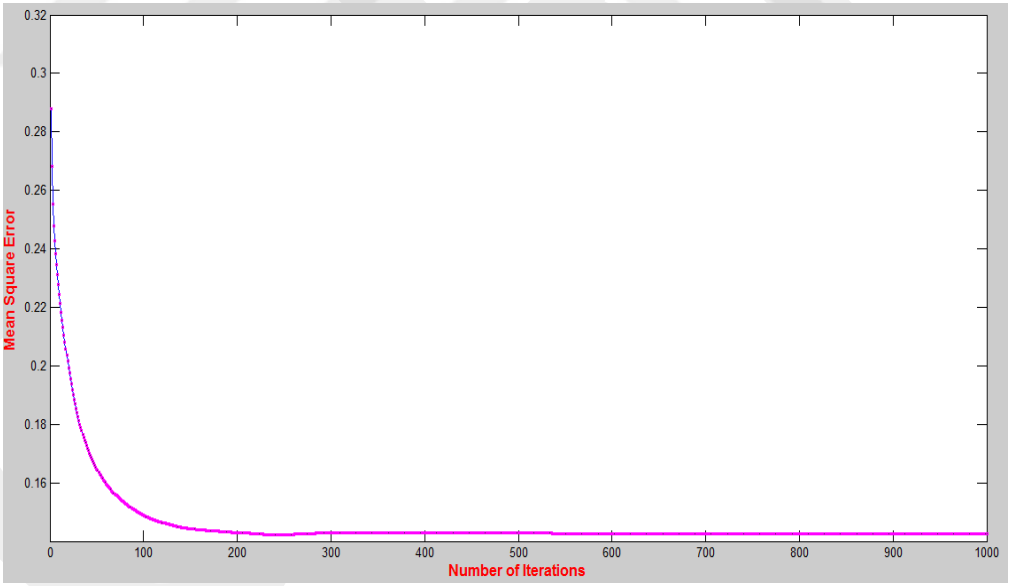

Figure 4.1 MSE variation for HNC-SDP model over IVY dataset

Moreover, in this work, MSE was considered as fitness value to be achieved while learning using the proposed heuristic model, and therefore, it can be understood that the RMSE error must be decreasing as per increase in generations, i.e., after every

generation the error would reduce and hence approaching towards optimal fitness value. Thus, as per genetic computing rule, with increase in generation, the error must be decreasing. The results obtained in this thesis (Figure 4.1, Figure 4.2, Figure 4.3 and Figure 4.4); it is confirmed that the proposed heuristic driven neuro-computing model (HNC-SDP) performs optimally in sync with expected performance towards defect prediction.

4.2.1.2 Test Case-2 ANT Dataset

Similar to the above stated IVY dataset simulation, the proposed HNC-SDP model was simulated over ANT1.7 PROMISE dataset. In this reference, the results obtained are given as follows:

Table 4.5 depicts the confusion matrix of ANT1.7 defect dataset before prediction or HNC-SDP execution. Similarly, the confusion metrics post HNC-SDP execution (which can be called as the proposed model) was obtained, and is portrayed in Table 4.6.

Table 4.5 Confusion matrix for ANT data before prediction

	Normal	**Fault/Defect**
Normal	578	0
Fault/Defect	166	9

Table 4.6 Confusion Matrix for ANT Data After Prediction

	Normal	**Fault/Defect**
Normal	578	0
Fault/Defect	157	9

In sync with above obtained confusion metrics (Table 4.6), the statistical performance results obtained are given in Table 4.7. The corresponding MSE variation over learning is given in Figure 4.2.

Table 4.7 Aggregate HNC-SDP Performance assessment for ANT data

Accuracy	Precision	F-Measure	Recall
0.8145	0.9343	0.8867	0.8438

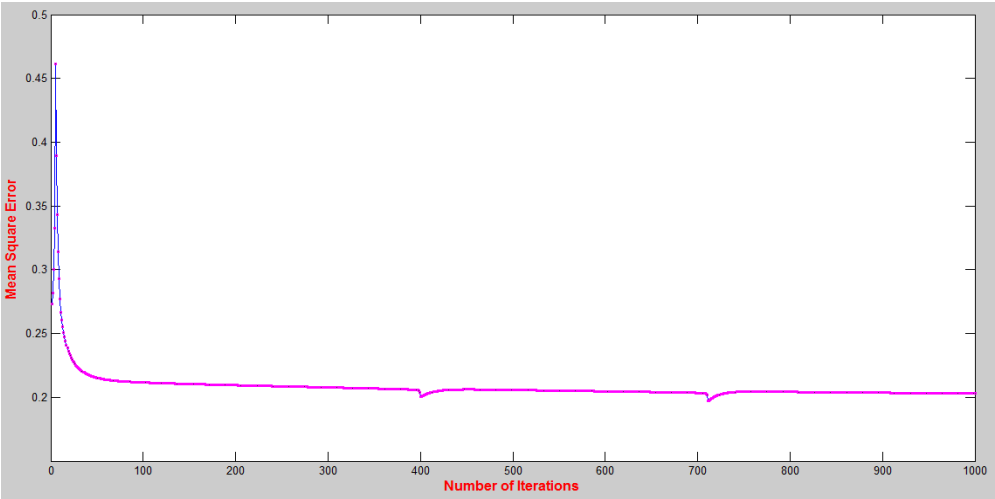

Figure 4.2 MSE variation for HNC-SDP model over ANT dataset

Observing the results (Table 4.7), it tends to be found that the proposed HNC-SDP model accomplishes accuracy of 81.4%, precision of 93.43, recall or sensitivity of 84.3% and F-Measure of 88.67% (with ANT1.7 dataset). This result confirms efficacy of the proposed model; though, its relative performance with other state-of-art methods is yet to be done (refer, Section 4.2.2).

4.2.1.3 Test Case-3 JEDIT Dataset

Table 4.8 displays the confusion matrix for JEDIT data prior to HNC-SDP execution. The confusion matrix recovered after HNC-SDP execution is shown in Table 4.9.

Table 4.8 Confusion matrix for JEDIT data prior to prediction

	Normal	**Fault/Defect**
Normal	481	0
Fault/Defect	11	0

Table 4.9 Confusion matrix for JEDIT data later of prediction

	Normal	**Fault/Defect**
Normal	481	0
Fault/Defect	10	1

In reference to the proposed HNC-SDP model execution, the statistical results for Table 4.9, is given in Table 4.10.

Table 4.10 addresses the overall efficacy of the suggested HNC-SDP for JEDIT data. Observing the results, one can find that the accuracy of the proposed HNC-SDP model over JEDIT dataset is almost 98%. On the other hand, various criteria, such as precision, recall and F-Measure values have been discovered as 100%, 100% and 98.97%, respectively. Moreover, the MSE variation too indicates superior learning which has resulted optimal performance, as depicted in Table 4.10.

Table 4.10 Cumulative HNC-SDP Performance evaluation for JEDIT data

Accuracy	**Precision**	**Recall**	**F-Measure**
0.9799	1	1	0.9897

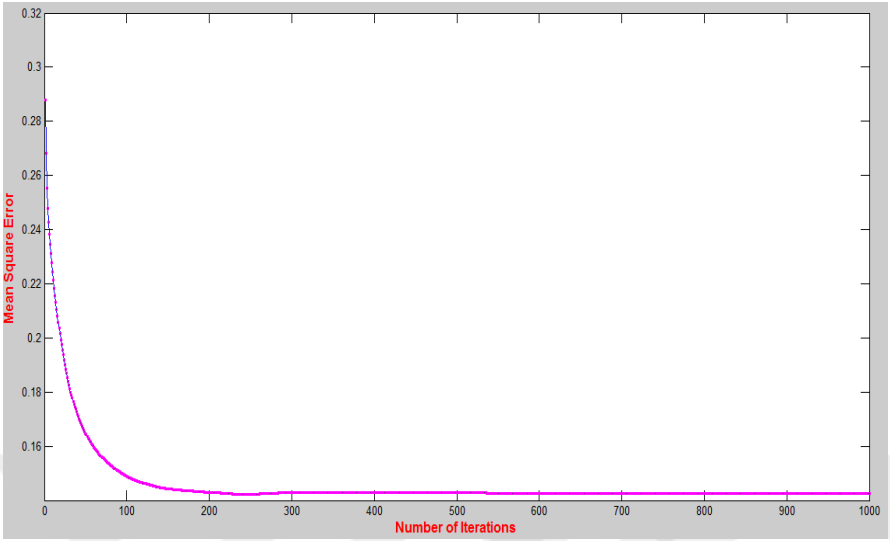

Figure 4.3 MSE variation for HNC-SDP model over JEDIT dataset

Test Case-4 CAMEL Dataset

Similar to the JEDIT datasets for defect prediction, the following tables (Table 4.11 and Table 4.12) represent the confusion matrix before and after HNC-SDP based SDP prediction, respectively (For CAMEL dataset).

Table 4.11 Confusion matrix for Camel data prior to prediction

	NON-FAULTY	**FAULTY**
Normal	777	0
Fault/Defect	188	0

Table 4.12 Confusion matrix for Camel data later prediction

	NON-FAULTY	**FAULTY**
Normal	770	7
Fault/Defect	172	16

Table 4.13 Cumulative HNC-SDP Performance evaluation for Camel data

Accuracy	Precision	Recall	F-Measure
0.8114	1	0.8102	0.8952

Observing the results (Table 4.13), the suggested HNC-SDP model may readily be accessed and achieves higher precision; though accuracy of 81% indicates exceedingly high non-linearity and hence low performance by the proposed HNC-SDP model. Yet, higher value of F-measure (89.5%) and recall (81.2%) shows that the proposed SDP model is capable to deliver reliable performance even over high non-linear inputs or training data. Interestingly, unlike Figure 4.1, Figure 4.2 and Figure 4.3, the error proneness has increased in case of CAMEL data set (Figure 4.4). Here the predominant reason is the non-uniform error distribution. The non-uniform error distribution in CAMEL data which is even high in size too, causes a fluctuation in error (MSE), but the robustness of the proposed HNC-SDP model can be observed here that just within few iterations, it retains the stabilized performance with reduced error and elevating trends towards optimal fitness value.

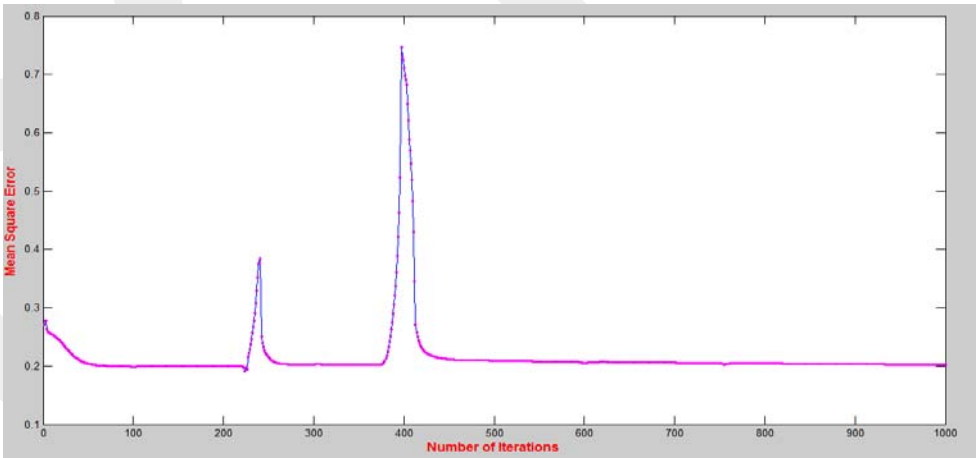

Figure 4.4 MSE variation for HNC-SDP model over CAMEL dataset

Undeniably, the swift convergence can enable any machine learning model to provide superior accuracy and hence the at hand SDP model (i.e., HNC-SDP), where it ensures

to deliver results before getting conserved or allied likelihood. As, conventional neuro-computing model undergoes local minima and convergence, this study focused on achieving prediction results before the neuro-computing model gets saturated. Considering this fact, this thesis developed HNC-SDP which is as heuristic driven neuro-computing model. In this method, after every generation the proposed or employed heuristic model (say improved genetic algorithm) ensured that the fitness value of the candidate chromosome or sub-solution increases. As, HNC-SDP considered inverse of MSE as fitness $\left(F_i = {}^{1}/_{E_i}\right)$, where E_i denotes the fitness value of the chromosome and F_i is the fitness value of that chromosome, the reduction in MSE (E_i) signifies better solution and allied learning. In this reference, observing Figure 4.1 to Figure 4.4., it can be found that the implementation of the proposed heuristic model helped ANN to alleviate any probable local minima and convergence. While, the use of SMOTE sampling followed by Min-Max normalization helped in alleviating any probability of over-fitting or data imbalance. These all efficacy ensured the proposed model achieving better learning and hence better results (Table 4.4, Table 4.7, Table 4.10 and Table 4.13).

The overall performance summary over the different input dataset (by the proposed HNC-SDP) model is given in Table 4.14.

Table 4.14 Summary of Intra-model performance assessment

Dataset	**Accuracy (%)**	**Precision (%)**	**F-Measure (%)**	**Recall (%)**
IVY1.7	88.35	99.36	93.80	88.83
ANT1.8	81.45	93.43	88.67	84.38
JEDIT	97.99	100.00	100.00	98.97
CAMEL	81.14	100.00	81.02	89.52

4.2.2 Inter-Model Assessment

In order to examine relative efficacy of the proposed machine learning model (i.e., HNC-SDP) model, at first its comparison was made with the other base algorithm like artificial neural network (ANN). Here, the only motive was to assess whether the inclusion of the proposed heuristic driven neuro-computing concept (HNC-SDP) serves superior results over the classical machine learning (i.e., ANN) or not. Similar to the proposed HNC-SDP model, the simulation of ANN was done over all four input datasets (i.e., ANT, IVY, JEDIT and CAMEL). The relative performance comparison is given in Figure 4.5, Figure 4.6, Figure 4.7 and Figure 4.8. When looking at the data, it can be seen that the average fault prediction accuracy of the suggested HNC-SDP model on opposite, the ANN based SDP models conveys 75.48% and subsequently the proposed framework beats the current and till most productive ANN model. Also, the proposed system with its best possible efficiency has exhibited defect prediction accuracy of 98%, precision of 100%, F-measure as 98.9% and recall efficiency as 100%. These all-performance parameters do affirm and justify robustness of the proposed HNC-SDP model towards software defect prediction purposes.

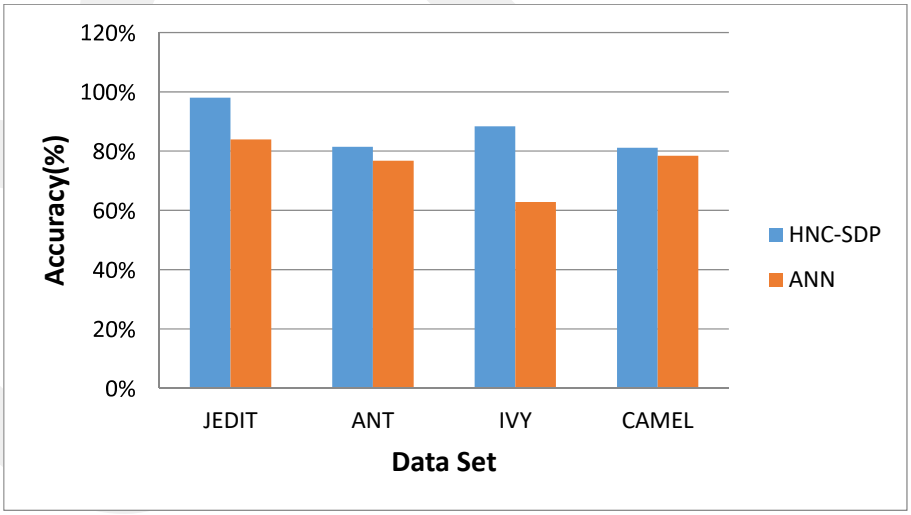

Figure 4.5 Comparison of the accuracy of HNC-SDP and ANN

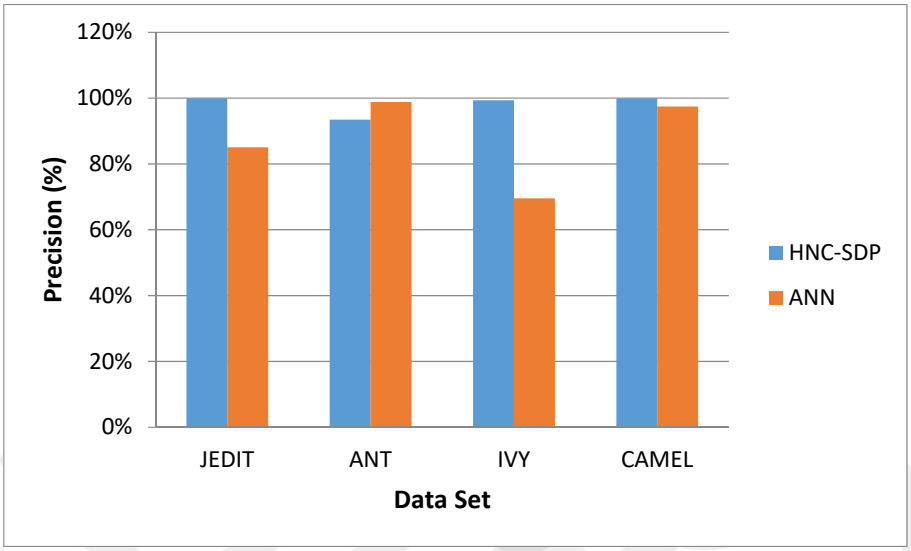

Figure 4.6 Comparison of the Precision of HNC-SDP and ANN

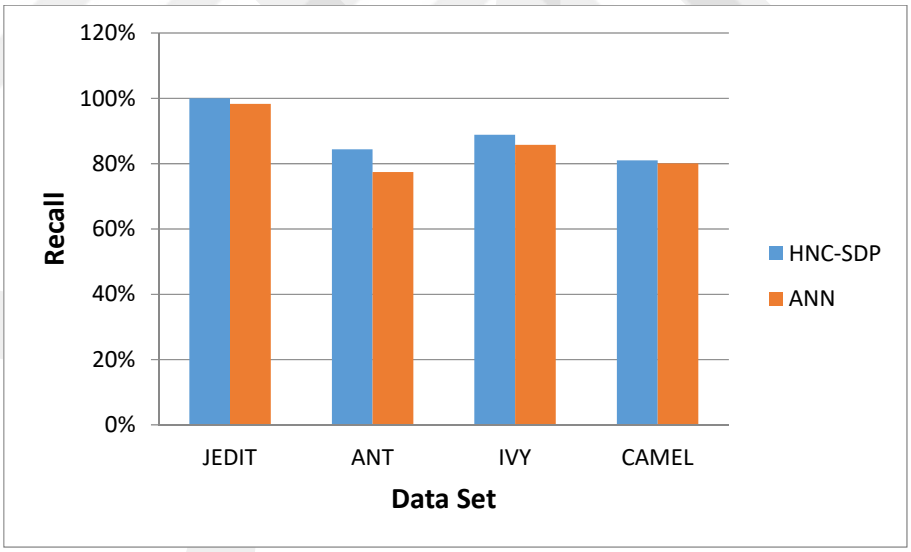

Figure 4.7 Comparison of the Recall of HNC-SDP and ANN

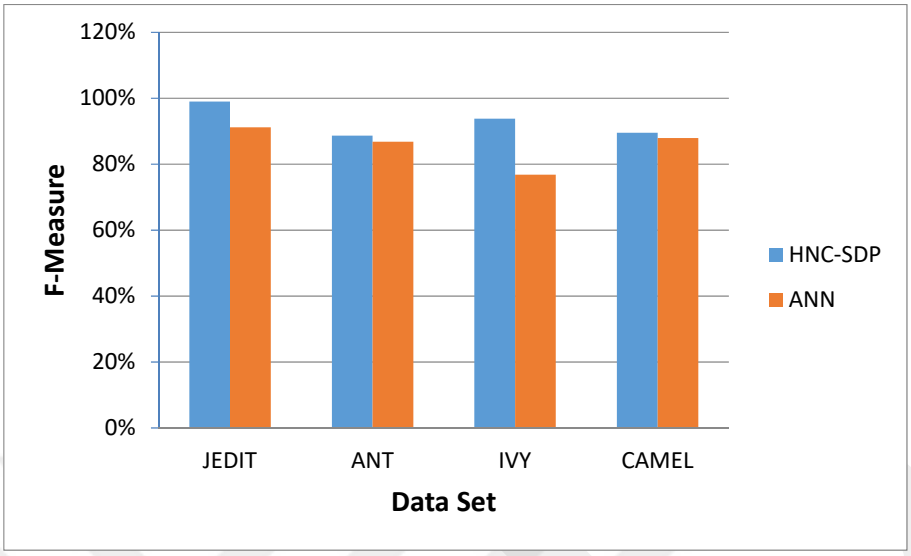

Figure 4.8 Comparison of the F-Measure of HNC-SDP and ANN

F-measure being harmonic means of precision and recall, with higher value in HNC-SDP states better performance for defect prediction, as compared to ANN algorithm. Furthermore, higher recall of the proposed HNC-SDP depicts that the proposed HNC-SDP model is higher sensitive as compared to ANN based SDP model.

In order to examine the effectiveness of the, proposed HNC-SDP model deeper in comparison to others state-of-art methods, qualitative study approach was considered. Different literatures discussing machine learning based SDP were studied and corresponding performance were examined. The relative performance analysis between the proposed HNC-SDP model and other existing approaches is given in Table 4.15.

Table 4.15 Various SDP strategies are compared in terms of performance

Reference	Machine Learning Techniques	Accuracy (%)	Precision (%)	F-Measure (%)
[86]	LLE-SVM	81.1	82.5	80.4
[86]	SVM	69.4	68.1	69.7

[87]	SVM	55.3	88.0	83.2
[88]	Natural Gas	94.2	-	-
[88]	Symbolic Regression	89.50	-	-
[88]	RBP-NN	80.0	-	-
[87]	LP	86.6	86.6	87.4
[87]	Naive Based	85.6	83.1	83.9
[89]	CPSO	69.2	67.6	-
[51]	T-SVM	75.8	84.1	80.9
[89]	GANN	73.4	81.6	-
[89]	AdaBoost	79.1	82.3	-
[90]	Random Forest	91.4	-	-
[91]	k-NN	91.8	-	-
[91]	C4.5	88.3	-	-
[91]	J 48	90.9	-	-
[91]	Levenberg-Marquardt	88.0	-	-
[89]	NNEP-Evolutionary	88.8	81.2	-
[92]	PSO	78.7	-	-
[88]	PSO-NN	97.7	-	-
Proposed	HNC-SDP	97.9	1	98.9

Observing the results (Table 4.15), It is simple to discover that the suggested HNC-SDP model performs superior over any other existing methods. Therefore, the proposed model can be robust towards any SDP task. The detailed discussion of the

overall research conclusion and allied inferences is given in the subsequent chapter (Chapter-5).

4.3 Summary

This chapter discusses the overall simulation results of the proposed HNC-SDP model, where the performance characterization was made in terms of intra-model as well as inter-model assessment. The overall results for all test cases (with the different input datasets), revealed that the proposed HNC-SDP model performs superior over other existing (state-of-art) machine learning based SDP methods including conventional neural network-based approaches. The depth assessment revealed that the suggested HNC-SDP model outperforms other existing machine learning based approaches when it comes to accuracy, precision, recall and F-Measure values. The general research conclusion and related aspects are discussed in depth in Chapter-5.

CHAPTER 5
CONCLUSION

The tremendous increase of software technology and related expectations among all socio-scientific and economic horizon has made it an inevitable need to serve varied purposes. However, across the application horizon, reliability of software has remained an uncompromisable need to ensure secure, safe, accurate processing or decision system. On the contrary, to meet upsurging demands software are designed with highly complex and humongous architecture, where human error or any fault probability can't be ignored. There are other events as well including exceedingly high complexity, aging, excessive reusability of free open software solution or components, code smells, refactoring etc. that increase the likelihood of software defect or fault. Such kind of faults can adversely impact the overall reliability of the system and hence can cause economic loss, infrastructure loss and even loss of life. To alleviate such issues, software defect prediction is an inevitable task. However, as stated, the contemporary software which are gigantic in size, performing manual class-by-class fault assessment is highly difficult and resource exhausting. To alleviate it, machine learning models can be of great significance. However, there are a number of limitations including class imbalance probability, local minima and convergence issues, over-fitting etc. that make majority of the classical machine learning models confined towards software defect prediction.

Considering above facts as motivation, in this dissertation a novel and robust heuristic driven neuro-computing model was developed for software defect prediction. Unlike other classical machine learning models, neuro-computing, especially Levenberg Marquardt Neural Network (LM-ANN), is considered to be more robust in terms of adaptive learning, which can be vital towards non-linear feature learning and hence defect data. However, similar to the other machine learning models, the likelihood of local minima and convergence could not be avoided due to exceedingly high weight estimation for 17 input features. Considering this fact, this research contributed a novel improved genetic algorithm, say heuristic model was developed to assist ANN

for adaptive weight estimation and update during learning. Here, the key purpose of heuristic model was to help LM-ANN gaining superior weight estimation, update and hence learning without undergoing any local minima and convergence problem. This as a result helped the proposed neuro-computing model to achieve higher accuracy than the classical neural network over targeted software fault datasets. In addition to the classifier or machine learning improvement, in this research the focus was made on feature engineering as well that helped alleviating any probability of class imbalance, over-fitting and convergence.

To assess efficacy or robustness of the proposed SDP model, in this dissertation, four different defect datasets named JEDIT, IVY. CAMEL and ANT were considered for the different case studies. To alleviate any possibility of class-imbalance the proposed HNC-SDP model at first performed resampling that improved quotient of the minority class samples to make learning superior. Subsequently, to alleviate any chance of over-fitting during non-linear pattern learning, HNC-SDP model applied Min-Max normalization that mapped each input data or feature in the range of 0 to 1. To be noted, as first of its kind solution, HNC-SDP model applied a total of 17 CKJM OOP (often called OOP-CK) software metrics as feature to perform defect prediction. Here, these 17 features were considered as the independent variable, while fault probability in each class was considered as the dependent variable. Thus, executing normalization of the input features, the normalized feature sets were projected for two-class arrangement utilizing HNC model. The HNC model that has been presented is based on the LM-ANN concept where for the deployed neural network, a total of 342 weights each iteration was really difficult. To alleviate this problem, HNC model applied an improved genetic algorithm or nature driven heuristic model that helped updating the weight parameters optimally. Thus, with the updated and optimised weight parameters the proposed HNC model performed two class classification and resulted classification results as “Normal Class” or “Defect or Fault Class”, and labelled them as 0 and 1, respectively. Thus, the proposed HNC-SDP model performed classification for each class and performed per-class or per-function defect prediction.

Simulating the proposed model over the different input datasets, this thesis revealed that unlike classical LM-ANN or neuro-computing model, the proposed heuristic

driven neuro-computing (HNC) model delivered superior performance when it comes to accuracy, precision, recall and F-Measure. The highest accuracy observed by the proposed HNC-SDP model was 98%, which was significantly higher than the classical neural network model. The depth analysis for intra-model comparison revealed that the proposed HNC-SDP model delivers almost 10% superior accuracy, precision, recall and F-Measure, signifying robustness over state-of-art method. This can be mainly contributed due to better learning and convergence problem alleviation. Moreover, HNC-SDP can be hypothesized to be more effective to alleviate false positive performance that resulted into higher accuracy over all input datasets (i.e., JEDIT, ANT, IVY and CAMEL). On the other hand, the inter-model assessment too revealed demonstrates the suggested HNC-SDP model outperforms the traditional model by a large margin in machine learning methods like ANN, Support Vector Machine (SVM), random forest, decision tree, regression methods etc. The overall perform affirms that the suggested HNC-SDP model can be vital towards real-time SDP tasks without imposing complex functional demand or requirements.

5.1 Future Work

Exploiting the intrinsic features, authors revealed that in large scale software the different features such as inheritance, polymorphism etc. cannot be effective as standalone feature for further classification and hence CK metrics seems to be the potential approach for automatic (machine learning-based) software defect prediction. Furthermore, applying GA algorithm to optimize the performance of the LM-ANN rather than traditional techniques. To be noted, in majority of the existing heuristic based (especially GA driven models), authors have applied crossover and mutation probability as 0.8, 0.2 or 0.6, 0.4 pair (say, pareto combination), respectively. However, this approaches often leads exceedingly rise in search space due to continuous accumulation of the candidate space (in each generation). Eventually, this itself raises a problem of convergence and local minima, which can greatly impact the overall performance. similar to the other machine learning models, the likelihood of local minima and convergence could not be avoided due to exceedingly high weight estimation for 17 input features.

REFERENCES

[1] Q. Li and H. Pham, "A generalized software reliability growth model with consideration of the uncertainty of operating environments," *IEEE Access*, vol. 7, pp. 84253–84267, 2019.

[2] S. Martínez-Fernández *et al.*, "Continuously assessing and improving software quality with software analytics tools: a case study," *IEEE access*, vol. 7, pp. 68219–68239, 2019.

[3] M. Mijač and Z. Stapić, "Reusability metrics of software components: survey," *Conf.Proceedings of the 26th Central European on Information and Intelligent Systems*, 2015, pp. 221–231.

[4] M. Lafi, J. W. Botros, H. Kafaween, A. B. Al-Dasoqi, and A. Al-Tamimi, "Code Smells Analysis Mechanisms, Detection Issues, and Effect on Software Maintainability," *Conf. IEEE Jordan International Joint on Electrical Engineering and Information Technology (JEEIT)* 2019, pp. 663–666.

[5] H. Liu, Q. Liu, Z. Niu, and Y. Liu, "Dynamic and automatic feedback-based threshold adaptation for code smell detection," *IEEE Trans. Softw. Eng.*, vol. 42, pp. 544–558, 2015.

[6] A. Baabad, H. B. Zulzalil, and S. B. Baharom, "Software architecture degradation in open source software: A systematic literature review," *IEEE Access*, vol. 8, pp. 173681–173709, 2020.

[7] F. Palomba, M. Zanoni, F. A. Fontana, A. De Lucia, and R. Oliveto, "Toward a smell-aware bug prediction model," *IEEE Trans. Softw. Eng.*, vol. 45, pp. 194–218, 2017.

[8] J. L. B. Justo, N. M. Araujo, and A. G. Garcia, “Software reuse and continuous software development: A systematic mapping study,” *IEEE Lat. Am. Trans.*, vol. 16, , pp. 1539–1546, 2018.

[9] C. Diwaker *et al.*, “A new model for predicting component-based software reliability using soft computing,” *IEEE Access*, vol. 7, pp. 147191–147203, 2019.

[10] M.-C. Chiang, C.-Y. Huang, C.-Y. Wu, and C.-Y. Tsai, “Analysis of a Fault-Tolerant Framework for Reliability Prediction of Service-Oriented Architecture Systems,” *IEEE Trans. Reliab.*, vol. 70, pp. 13–48, 2020.

[11] S. Maggo and C. Gupta, “A machine learning based efficient software reusability prediction model for java based object oriented software,” *Int Journal Inf. Technol. Comput. Sci.*, vol. 6, pp. 1–12, 2014.

[12] N. E. Fenton and M. Neil, “A critique of software defect prediction models,” *IEEE Trans. Softw. Eng.*, vol. 25, pp. 675–689, 1999.

[13] N. Padhy, R. P. Singh, and S. C. Satapathy, “Enhanced evolutionary computing based artificial intelligence model for web-solutions software reusability estimation,” *Cluster Comput,* vol. 22, pp. 9787–9804, 2019.

[14] A. C. Framework, “Supplement to IEEE Standard for Information Technology-Software Reuse-Data Model for Reuse Library Interoperability: Asset Certification Framework,” *IEEE Standard*, vol. 1, p. 60 ,1997.

[15] J. Tian, “Quality-evaluation models and measurements,” *IEEE Softw.*, vol. 21, pp. 84–91, 2004.

[16] K. Henningsson, “A fault classification approach to software process

improvement.” Blekinge Tekniska Högskola, 2005.

[17] B. Clark and D. Zubrow, “How good is the software: a review of defect prediction techniques,” in *Software Engineering Symposium Conf.* 2001,pp. 1–35.

[18] S. McConnell, “Gauging software readiness with defect tracking,” *IEEE Softw.*, vol. 14, p. 136, 1997.

[19] S. Chulani and B. Boehm, “Modeling software defect introduction and removal,” London ,1999.

[20] B. Bw, “Software Engineering Economics,” *IEEE Transactions on Software Engineering,* Vol.40, pp.4-21,1984.

[21] C. Jones, “Programming defect removal,” *Proceedings, Guid.*, vol. 40, 1975.

[22] T.-J. Yu, V. Y. Shen, and H. E. Dunsmore, “An analysis of several software defect models,” *IEEE Trans. Softw. Eng.*, vol. 14, pp. 1261–1270, 1988.

[23] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object oriented design,” *IEEE Trans. Softw. Eng.*, vol. 20, pp. 476–493, 1994.

[24] Y. Liu, T. M. Khoshgoftaar, and N. Seliya, “Evolutionary optimization of software quality modeling with multiple repositories,” *IEEE Trans. Softw. Eng.*, vol. 36, pp. 852–864, 2010.

[25] Q. Song, M. Shepperd, M. Cartwright, and C. Mair, “Software defect association mining and defect correction effort prediction,” *IEEE Trans. Softw. Eng.*, vol. 32, pp. 69–82, 2006.

[26] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, “Benchmarking

classification models for software defect prediction: A proposed framework and novel findings," *IEEE Trans. Softw. Eng.*, vol. 34, pp. 485–496, 2008.

[27] J. C. Munson and T. M. Khoshgoftaar, "The detection of fault-prone programs," *IEEE Trans. Softw. Eng.*, vol. 18, p. 423, 1992.

[28] T. Mitchell, October 1, 1997 "Machine learning," (2nd edition),1997.

[29] N. Ohlsson, M. Helander, and C. Wohlin, "Quality improvement by identification of fault-prone modules using software design metrics," *Sixth International Conf. on Software Quality*, 1996, pp. 1–13.

[30] D. Rodríguez, J. C. Riquelme, R. Ruiz, and J. S. Aguilar-Ruiz, "Searching for rules to find defective modules in unbalanced data sets," *International Conf.in Symposium Search on Based Software Engineering*, 2009, pp. 89–92.

[31] C. Catal and B. Diri, "Software defect prediction using artificial immune recognition system," *IASTED international multi- Conf.software engineering*, 2007,pp. 285–290.

[32] D. J. Drown, T. M. Khoshgoftaar, and N. Seliya, "Evolutionary sampling and software quality modeling of high-assurance systems," *IEEE Trans. Syst. Man, Cybern. A Syst. Humans*, vol. 39, pp. 1097–1107, 2009.

[33] J. Wang, B. Shen, and Y. Chen, "Compressed C4. 5 models for software defect prediction," *International Conf. on Quality Software*, 2012, pp. 13–16.

[34] Y. Chen, X. Shen, P. Du, and B. Ge, "Research on software defect prediction based on data mining," *International Conf. on Computer and Automation Engineering (ICCAE)*, 2010, pp. 563–567.

[35] B. Li, B. Shen, J. Wang, Y. Chen, T. Zhang, and J. Wang, "A scenario-based approach to predicting software defects using compressed C4. 5 model," *IEEE 38th Annual Computer Software and Applications Conf.*, 2014, pp. 406–415.

[36] T. Marwala, "Probabilistic fault identification using vibration data and neural networks," *Mech. Syst Journal. Signal Process.*, vol. 15, pp. 1109–1128, 2001.

[37] T. YAIRI, N. ISHIHAMA, Y. KATO, K. HORI, and S. NAKASUKA, "Anomaly detection method for spacecrafts based on association rule mining," *Journal Sp. Technol. Sci.*, vol. 17, pp. 1_1-1_10, 2001.

[38] A. G. Koru and H. Liu, "An investigation of the effect of module size on defect prediction using static measures," in *Proceedings Conf,on Predictor models in software engineering*, 2005, pp. 1–5.

[39] D. Zhang and J. J. P. Tsai, "Machine learning and software engineering," *Softw. Qual Journal*, vol. 11, pp. 87–119, 2003.

[40] V. U. B. Challagulla, F. B. Bastani, and I.-L. Yen, "A unified framework for defect data analysis using the mbr technique," *IEEE International Conf. on Tools with Artificial Intelligence (ICTAI'06)*, 2006, pp. 39–46.

[41] A. Trendowicz and T. Punter, "Quality modeling for software product lines," *Conf , in Object-Oriented Software Engineering*, 2003,pp.7.

[42] T. M. Khoshgoftaar, B. Cukic, and N. Seliya, "Comparative study of the impact of underlying models on module-Order model performances," *IEEE International Symposium on Software Metrics*, vol.8, p. 161,2002

[43] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Unsupervised learning for expert-based software quality estimation.," IEEE international *Conf.* on High

assurance systems engineering, 2004, pp. 149–155.

[44] V. U. B. Challagulla, F. B. Bastani, I.-L. Yen, and R. A. Paul, "Empirical assessment of machine learning based software defect prediction techniques," *Int Journal Artif. Intell. Tools*, vol. 17, pp. 389–400, 2008.

[45] M. Shepperd and G. Kadoda, "Comparing software prediction techniques using simulation," *IEEE Trans. Softw. Eng.*, vol. 27,pp. 1014–1022, 2001.

[46] C. Mair *et al.*, "An investigation of machine learning based prediction systems," *Journal Syst. Softw.*, vol. 53, pp. 23–29, 2000.

[47] K. El Emam, S. Benlarbi, N. Goel, and S. N. Rai, "Comparing case-based reasoning classifiers for predicting high risk software components," *Journal Syst. Softw.*, vol. 55, pp. 301–320, 2001.

[48] I. Watson and F. Marir, "Case-based reasoning: A review," *Knowl. Eng. Rev.*, vol. 9, pp. 327–354, 1994.

[49] T. M. Khoshgoftaar, L. A. Bullard, and K. Gao, "Detecting outliers using rule-based modeling for improving CBR-based software quality classification models," in *International Conf. on Case-Based Reasoning*, 2003, pp. 216–230.

[50] M. J. Khan, S. Shamail, M. M. Awais, and T. Hussain, "Comparative study of various artificial intelligence techniques to predict software quality,"*IEEE International Multitopic Conf.*, 2006, pp. 173–177.

[51] A. Chug and S. Dhall, "Software defect prediction using supervised learning algorithm and unsupervised learning algorithm," *Turkish Journal* of Physiotherapy and Rehabilitation, vol. 12, pp. 2429– 2436, 2013.

[52] T. Wang and W. Li, “Naive bayes software defect prediction model,” *International Conf*. on Computational Intelligence and Software Engineering, 2010 ,pp. 1–4.

[53] P. Singh and S. Verma, “An investigation of the effect of discretization on defect prediction using static measures,”*International Conf*. on Advances in Computing, Control, and Telecommunication Technologies, 2009, pp. 837–839.

[54] T. P. Pushphavathi, V. Suma, and V. Ramaswamy, “A novel method for software defect prediction: Hybrid of FCM and random forest,” *International Conf*. on Electronics and Communication Systems (ICECS), 2014, pp. 1–5.

[55] H. Zhang and X. Zhang, “Comments on" data mining static code attributes to learn defect predictors"," IEEE Trans. Softw. Eng., vol. 33, pp. 635–637, 2007.

[56] B. Shuai, H. Li, M. Li, Q. Zhang, and C. Tang, “Software defect prediction using dynamic support vector machine,” Ninth *International Conf*. on Computational Intelligence and Security, 2013, pp. 260–263.

[57] E. Ardil, E. Ucar, and P. S. Sandhu, “Software maintenance severity prediction with soft computing approach,” Int. Sch. Sci. Res. Innov. Proc. World Acad. Sci. Eng. Technol, vol. 3, pp. 253–258, 2009.

[58] Z. Jianhong, P. S. Sandhu, and S. Rani, “A neural network based approach for modeling of severity of defects in function based software systems,” *International Conf. on Electronics and Information Engineering*, 2010, pp. 12-568.

[59] D. Yuan and C. Zhang, “Evaluation strategy for software reliability based on ANFIS,” *International Conf. on Electronics, Communications and Control (ICECC)*, 2011, pp. 3738–3741.

[60] M. E. R. Bezerra, A. L. I. Oliveira, and S. R. L. Meira, “A constructive rbf neural network for estimating the probability of defects in software modules,” *International Joint Conf. on Neural Networks*, 2007, pp. 2869–2874,.

[61] G. D. Boetticher, T. Menzies, T. Ostrand, and G. H. Ruhe, “4th international workshop on predictor models in SE (PROMISE 2008),”*İnternational Conf. on Software engineering*, 2008, pp. 1061–1062.

[62] R. Fujimaki, T. Yairi, and K. Machida, “An approach to spacecraft anomaly detection problem using kernel feature space,” *ACM SIGKDD İnternational Conf. on Knowledge discovery in data mining*, 2005,pp. 401–410.

[63] T. Mende and R. Koschke, “Revisiting the evaluation of defect prediction models,” *International Conf. on Predictor Models in Software Engineering*, 2009, pp. 1–10.

[64] S. Kanmani, V. R. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai, “Object-oriented software fault prediction using neural networks,” *Inf. Softw. Technol.*, vol. 49, pp. 483–492, 2007.

[65] G. J. Pai and J. B. Dugan, “Empirical analysis of software fault content and fault proneness using Bayesian methods,” *IEEE Trans. Softw. Eng.*, vol. 33, pp. 675–686, 2007.

[66] Y. Singh, A. Kaur, and R. Malhotra, “Empirical validation of object-oriented metrics for predicting fault proneness models,” *Softw. Qual Journal* vol. 18, pp. 3–35, 2010.

[67] R. Malhotra and A. Jain, “Fault prediction using statistical and machine learning methods for improving software quality,” *Journal Inf. Process. Syst*,vol. 8,pp. 241–262, 2012.

[68] A. Janes, M. Scotto, W. Pedrycz, B. Russo, M. Stefanovic, and G. Succi, "Identification of defect-prone classes in telecommunication software systems using design metrics," *Inf. Sci. (Ny).*, vol. 176, pp. 3711–3734, 2006.

[69] P. Tomaszewski, J. Håkansson, H. Grahn, and L. Lundberg, "Statistical models vs. expert estimation for fault prediction in modified code–an industrial case study," *Journal Syst. Softw.,* vol. 80, pp. 1227–1238, 2007.

[70] R. Subramanyam and M. S. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects," *IEEE Trans. Softw. Eng.*, vol. 29, pp. 297–310, 2003.

[71] V. Basili, B. Murphy, and N. Nagappan, "The Influence of Organizational Structure on Software Quality" ACM *Conf*, on Computer Supported Cooperative,2011, pp. 143–150,.

[72] T. Systä, *Static and dynamic reverse engineering techniques for Java software systems*. Tampere University Press, 2000.

[73] A. Schröter, T. Zimmermann, and A. Zeller, "Predicting component failures at design time,"*ACM/IEEE international symposium on Empirical software engineering*, vol. 1, pp. 18–27, 2002.

[74] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Trans. Softw. Eng.*, vol. 31, pp. 340–355, 2005.

[75] M.-H. Tang, M.-H. Kao, and M.-H. Chen, "An empirical study on object-oriented metrics," *international software metrics symposium Conf*, 1999, pp. 242–249.

[76] L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter, “Exploring the relationships between design measures and software quality in object-oriented systems,” *Journal Syst. Softw.*, vol. 51,pp. 245–273, 2000.

[77] P. Yu, T. Systa, and H. Muller, “Predicting fault-proneness using OO metrics. An industrial case study,” *european Conf. on software maintenance and reengineering*, 2002, pp. 99–107.

[78] Y. Zhou and H. Leung, “Empirical analysis of object-oriented design metrics for predicting high and low severity faults,” *IEEE Trans. Softw. Eng.*, vol. 32, pp. 771–789, 2006.

[79] 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 Trans. Softw. Eng.*, vol. 33, pp. 402–419, 2007.

[80] M. Alshayeb and W. Li, “An empirical study of system design instability metric and design evolution in an agile software process,” *Journal Syst. Softw.*, vol. 74, pp. 269–274, 2005.

[81] R. Harrison, S. J. Counsell, and R. V Nithi, “An evaluation of the MOOD set of object-oriented software metrics,” *IEEE Trans. Softw. Eng.*, vol. 24, pp. 491–496, 1998.

[82] V. R. Basili, L. C. Briand, and W. L. Melo, “A validation of object-oriented design metrics as quality indicators,” *IEEE Trans. Softw. Eng.*, vol. 22, pp. 751–761, 1996.

[83] R. Moser, W. Pedrycz, and G. Succi, “A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction,” *Conf. on Software*

engineering, 2008, pp. 181–190.

[84] F. Rosenblatt, "Principles of Neurodynamics Spartan," *New York*, 1962.

[85] J. Hertz, A. Krogh, and R. G. Palmer, "Introduction to the Theory of Neural Computation", Boulder, CO 80309-0430, USA :CRC Press,1991,pp.6-11.

[86] C. Shan, B. Chen, C. Hu, J. Xue, and N. Li, "Software defect prediction model based on LLE and SVM," Communications Security Conf, 2014, pp.1-6.

[87] Y. Xia, G. Yan, X. Jiang, and Y. Yang, "A new metrics selection method for software defect prediction," *IEEE International Conf. on Progress in Informatics and Computing*, 2014, pp. 433–436.

[88] A. Shrivastava and V. Shrivastava, "A hybrid model of soft computing technique for software fault prediction," *Int. Journal Curr. Eng. Technol.*, vol. 4, pp. 2511–2518, 2014.

[89] R. Malhotra, N. Pritam, and Y. Singh, "On the applicability of evolutionary computation for software defect prediction," *International Conf. on Advances in Computing, Communications and Informatics (ICACCI)*, 2014, pp. 2249–2257.

[90] M. M. Askari and V. K. Bardsiri, "Software defect prediction using a high performance neural network," *Int. Journal Softw. Eng. Its Appl.*, vol. 8, pp. 177–188, 2014.

[91] M. Singh and D. S. Salaria, "Software defect prediction tool based on neural network," *Int Journal Comput. Appl.*, vol. 70, 2013.

[92] R. Verma and A. Gupta, "Software defect prediction using two level data pre-processing," *International Conf. on Recent Advances in Computing and Software Systems*, 2012, pp. 311–317.