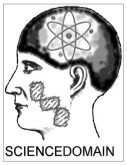
British Journal of Mathematics & Computer Science
16(5): 1-12, 2016, Article no.BJMCS.26337
ISSN: 2231-0851

SCIENCEDOMAIN *international*
www.sciencedomain.org



Defect Prediction Framework Using Neural Networks for Software Enhancement Projects

Vipul Vashisht[1*], Manohar Lal[1] and G. S. Sureshchandar[2]

[1]SOCIS, IGNOU, India.
[2]ASQ, Chennai, India.

Authors' contributions

This work was carried out in collaboration among all authors. Author VV conceived and designed the work and wrote the first draft of the manuscript. Authors ML and GSS helped perform the analysis of study with constructive discussions. All authors read and approved the final manuscript.

Article Information

DOI: 10.9734/BJMCS/2016/26337
Editor(s):
(1) Dijana Mosic, Department of Mathematics, University of Nis, Serbia.
Reviewers:
(1) Derya Sevim Korkut, Duzce University, Duzce, Turkey.
(2) Ankur Singh Bist, UPTU, India.
Complete Peer review History: http://sciencedomain.org/review-history/14542

Original Research Article

Received: 11th April 2016
Accepted: 26th April 2016
Published: 9th May 2016

Abstract

So far, various approaches have been proposed for effective and accurate prediction of software defects, yet most of these approaches have limited adoption in practice. The objective of this paper is to provide a framework which is expected to be more user-friendly, effective and acceptable for predicting the defects in multiple phases across software enhancement projects. This communication describes a process of applying computational intelligence technologies, in particular neural networks in formulating defect prediction models early in the software development life cycle. A series of empirical experiments are carried out based on input and output measures extracted from 50 'real world' project subsystems. In order to increase the adoption and make the prediction framework easily accessible to project managers, a graphical user interface (GUI) based tool has been designed and implemented that allows input data to be fed easily.

The proposed framework uses historical data for training model and as a result provides a defect range (minimum, maximum) based output instead of a definite defect count based output. This is done in view of the fact that exact-count prediction has less probability of being correct as compared to range based predictions. The defect predictions can be used for taking informed decisions including prioritizing software testing efforts, planning additional round of code reviews, allocating human and computer

**Corresponding author: E-mail: vipulvashisht@gmail.com;*

resources, planning for risk mitigation strategy and other corrective actions. The claim of effectiveness of proposed framework is established through results of a comparative study, involving the proposed framework and some well-known models for software defect prediction.

Keywords: Software defect; software defect prediction model; Neural Network (NN); quality management.

1 Introduction

The constantly evolving technological infrastructure presents a great challenge for developing better, faster and cost effective software systems. The increasing complexity of software products and projects has been constantly pushing software organizations to improve product quality and performance. Most organizations have business goals of customer satisfaction and profitable growth, which are being met through increasing use of software systems. In organizations, defect count is most commonly used as one of the major indicator of product quality. A higher defect count may not only affect the planned cost and schedule but may also result in losing the customer base. Hence planning for reducing defect count during software development can bring significant business benefits. Most experts agree to the fact that it is always better to prevent defects or detect them earlier in software development life cycle rather than to let the end customer find them.

In the current context, developing defect free software is a daunting task, specially, when software is being developed for problems with increasing complexity. However, occurrences of certain defects are inevitable in spite of all measures planned by the organizations. In order to control and reduce defect injection in software engineering processes, organizations not only have to plan huge budgets for time and resources but also need to plan for appropriate defect prediction model [1]. The Software Capability Maturity Model Integration (SW-CMMI) framework from CMMI Institute provides a set of requirements that organizations can use in setting up the software process used to control software development process and guide organizations in high performance operations. The CMMI framework highlights use of defect prediction model as one of the high maturity practices. In this regard, IT organizations must make use of quantitative techniques like control charts and prediction models to showcase the process improvements while planning for CMMI L5 appraisal from CMMI Institute [2,3]. Software defect prediction framework once implemented is used as effective tool by organizations for the purpose of identifying parts/phases of a software life cycle, requiring increased focus before release and taking necessary corrective and preventive actions towards reducing defect leakage.

1.1 Software enhancement project life cycle

A complex software development project would typically consist of phases such as, Requirement gathering, Design, Construction (including Coding and Unit Testing), System Testing, User Acceptance Testing, Implementation, and Post Implementation support. In the current business context, there are ever changing business needs, and as a consequence, frequent changes to an already implemented project have become an integral part of the process. These changes are made and controlled through due approvals from select teams usually termed as Change Advisory Board (CAB) constituted by the organization. Changes are generally approved on the basis of new requirements that have touch point(s) to the code of the already existing software. Depending on factors such as changes in size (function points/line of codes), the priority and the criticality, the project team generally decides whether the new change should be considered as a production support ticket or an enhancement by itself. In view of the fact that software enhancements are generally of smaller duration, these are termed as mini or short projects. The enhancement life cycles generally covers phases such as Requirement Gathering, Impact Analysis, Construction, Testing with UAT and Post Implementation. Effort spent under each of the software enhancement includes production effort, review effort and rework effort. Both effort and defects are interrelated by the fact that the production effort yields the number of defects injected, the review effort yields the number of defects detected and the rework effort removes the defects so detected. It is a well known fact that there is a relationship between functionality

enhancement and software defects distribution [4]. A prediction model generally predicts the number of defects as against the amount of effort for each of the phases aforementioned. Enhancement life cycle methodology represents improvements to existing software in terms of functionality / technologies. This enhancement methodology is suitable for ongoing maintenance of large applications. These projects carry out identified enhancements as a part of periodic releases and / or individual basis depending on priority and customer preferences.

The enhancement methodology is most suited to projects in the following scenarios:

- Small changes to a large application
- Significantly changes in core architecture and functionality
- Simpler requirements / requirement changes
- Small changes in functional and technology upgrade

In previous paper [5], the authors have presented a neural network based framework to predict defects in large software projects based on *waterfall lifecycle*. The overall duration of most of software enhancement project is quite small as compared to waterfall lifecycle based projects. In this communication, three distinct phases (Requirement Gathering, Construction and Testing) from the software enhancement projects have been considered for designing the framework. In view of the fact that the effort and duration for analysis and design phase plays a minimal role, hence, the defect prediction for these phases is not taken into consideration.

It has also been observed that relevant research work in the field past is focused more towards software development as compared to software enhancements or software maintenance [6]. Unfortunately, there is shortfall of appropriate defect prediction models for software enhancement projects. To address this problem, we propose a defect prediction model for software enhancement projects, justifying the investigations reported through this communication.

The paper is organized as follows. Section II provides a brief overview of Neural Networks, Section III reviews the existing literature on the subject, Section IV describes the proposed framework and discusses the results as obtained through use of the proposed defect prediction framework vis-à-vis some other relevant models/frameworks, and finally Section V concludes the presented work.

2 Computational Intelligence Technologies

During the previous decade, there has been increased integration between the fields of software engineering and computational intelligence (Cl). Where, the CI includes mature technologies of fuzzy logic, neural networks, genetic algorithms, genetic programming, rough sets and hybrid systems that combine two or more of these individual technologies like ANFIS. The computational intelligence area provides a unique opportunity of incorporating these technologies to address various software engineering problems. The main objective of incorporating Cl technologies into the various SDLC phases is to address the issues arising due to imprecise measurement and uncertainty of information [7,8]. The next section discusses about the structure and functions of a Neural Network.

2.1 Neural networks

Artificial Neural Network (ANN) approach is inspired by the human brain networks, which is a network of about 100 billion neurons, each neuron being connected, on the average, to about 1000 other neurons. A neuron is basic constituent of human brain, a sort of elementary processor having small local memory and capable of localized information processing. In Fig. 1, the leftmost layer in this network is called the input layer which consists of neurons called input neurons, through which inputs from the environment are received by the ANN. The rightmost or output layer contains the output neurons through which output to the environment are delivered by ANN. The middle layer (s) is called a hidden layer in which the actual

processing by ANN takes place. The signal or input given to one neuron is passed to all the neurons to which it is connected in fractions equivalent to the weight between these neurons. Each neuron-to-neuron connection has a variable weight quantifying the connection strength. Each neuron calculates its output based on a function which can be sigmoid, step or some such suitable function. In the first phase, the neural network performs learning by finding a vector of interconnection weights that minimizes its error on the data set used for training that has known values of inputs and corresponding outputs. In the next phase, after selection of the connection weights, the network predicts the output values for data having known inputs and unknown outputs.

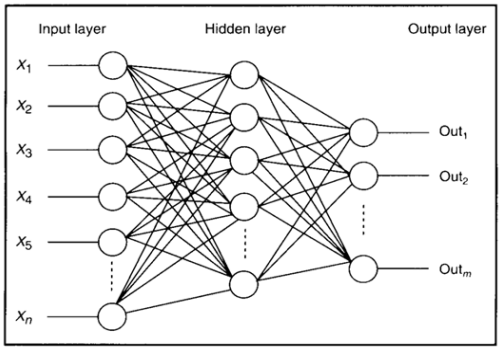


Fig. 1. Neural network [5]

Depending upon the pattern (architecture) of the connection, artificial neural network can be classified into two categories, feed-forward networks and recurrent (or feedback) networks. In feed-forward networks, graphs have no loops and the output from one layer is used as input to the next layer. In feedback networks, there are loops providing feedback connection to input layer.

One of the major advantages of neural networks over traditional expert systems is their ability to automatically learn from examples. ANNs have the ability to learn underlying rules (like input-output relationships) from the given collection of representative examples. A neural network learns patterns by adjusting its weights. When the neural network is properly trained, it can give correct, or nearly correct, answers for not only the sample patterns, but also for new similar patterns [9,10].

3 Related Literature Review

Software defect prediction is an active research area in field of software engineering. Researchers have proposed new defect prediction algorithms and/or new metrics to effectively predict defects. The historical data of software systems is a valuable asset used for research ranging from software design to software development, software maintenance, software testing, etc. In view of the fact that each defect prediction model has its own set of advantages and disadvantages, it is difficult to find most appropriate model for a particular type of project scenario, especially in view of the fact that every software project tends to be unique.

Neural networks have been found to be effective in situations where data relationships may not be known, as normally happens in the case of software defect prediction. It was observed during literature review that neural network based framework for modeling defect prediction has been successful in following application areas:

- Diverse fields range from autonomous vehicle control [11].
- Financial risk analysis to handwriting recognition [12].
- Dynamic software reliability modeling [13].

- Applying neural networks in software effort estimation [14,15].
- Software metrics models [16,17,18,19,20,5].

Prior to conducting the experiments, it is necessary to decide upon appropriate computational intelligence approach. It was decided to use neural network paradigm for creating defect prediction model framework for software enhancement project. This decision was based upon the fact that the relationship between effort and defect data is quite complex and is generally difficult to represent in functional or near functional form. The decision is further strengthened by the author's previous experience with using neural networks to model software defect prediction tool for Java waterfall life cycle based software projects [5].

In past, McCabe [21] and Halstead [22] work based on metrics have been commonly used to describe the attributes of each software module (i.e. the unit of functionality of source code). General principles/approaches/steps which have been found useful so far in handling the difficult task of software defect prediction, along with the relevant literature, are summarized below:

- In [23], an enhanced Multilayer Perceptron Neural Network based technique has been used for defect prediction. Comparative analysis of defect proneness predictions was performed using dataset from NASA MDP (Metrics Data Program). The results from proposed MLP neural network model were better when compared with existing techniques like Random Tree, classification and regression trees (CART) algorithm, and Bayesian logistic regression.
- In [24], Adaptive Resonance Neural Network having 29 input nodes and two output nodes is designed for the purpose of defect prediction in software programs. PROMISE dataset is used to train the network. The results showed that recall (true positive) rate is improved in predicting whether a module is defective or not [25].
- In [26], a software reliability modeling approach in terms of the predictive quality and the quality of fit is described using neural and regression analysis techniques. The data set has been taken from an Ada development environment for the command and control of a military data link communication system (CCCS). Results showed that the neural network based model has smaller standard error and is superior to traditional regression based techniques.
- In [5], neural network based defect prediction model has been successfully used for predicting defects across Java based projects following waterfall life cycle. The tests conducted for 15 projects showed accuracy close to 90 %.
- In [27], neural network based tool using Levenberg-Marquardt (LM) algorithm is used for software defects. The PROMISE repository dataset uses the CKOO (Chidamber and Kemerer Object-Oriented) metrics. The results showed that neural network based algorithm provides better accuracy (88.09%) as compared to each of polynomial function-based neural networks (pF-NNs), linear function-based neural network (lf-NN) and quadratic function-based neural network (qf-NN) respectively.

This paper describes a neural network based framework for formulating models for defect prediction early in the software life cycle. For the purpose, a series of empirical experiments is conducted based on input and output measures extracted from 'real world' projects. The experiments establish the efficacy and superiority of the approach. Next section describes the proposed framework.

4 The Proposed Framework

As mentioned earlier, the objective of this study is to develop prediction for forecasting the defects in software enhancement projects. In this section, first the assumptions made about the proposed framework have been explained. Then the structure of the proposed framework and functions of major components of the framework are described. The results and other related issues are discussed in the next section.

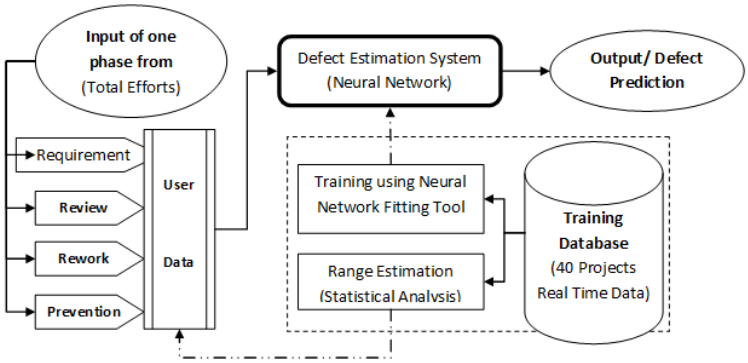


Fig. 2. Model framework design

The experiments reported here involve data set taken from 50 real projects from a software organization. Out of this dataset, 40 projects are used for training the model and the rest 10 projects data are used to validate the accuracy of the model. The actual defect data is taken from completed software enhancement projects. This historical data has served as a training data to build the proposed framework (refer Fig. 2) and then the neural network so obtained is used to predict the defects for all new projects.

While executing software projects, estimated effort is the primary determinant for arriving at the overall development cost and project schedule. In this communication, the defect prevention effort, review effort and rework effort have been considered along with the production effort. In view of the fact that as voluntary effort towards defect prevention activities increases, there is a considerable decrease in involuntary costs of rework leading to overall better quality [24,28]. The effort estimation for all software enhancement projects considered in this paper has been done using standard program complexity estimation technique [29].

4.1 Structure of the proposed neural network

A feed-forward network with sigmoid (hidden and linear output neurons) is used to formulate the system. The network is trained with scaled conjugate gradient back-propagation (training).

Defect prediction system consists of three parallel neural networks with different configurations and parameters for each sub phase. Only first phase, that is, Requirement Gathering phase is having 10 hidden layers. The architectural view of Neural Network for requirement gathering phase is shown in Fig. 3.

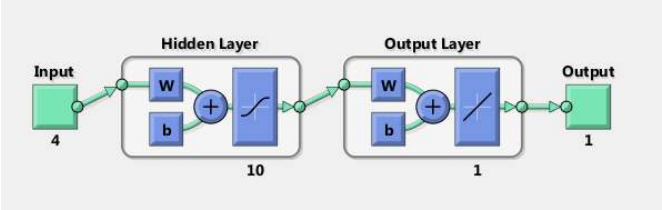


Fig. 3. Model framework design

The decision on number of hidden layers to be used is done while considering the need of optimizing the regression value for attaining the best performance. Despite the fact that use of more neurons require more computation and also such use leads to over fitting the data yet, at the same time, it allows the network to solve more complicated problems [30].

Input data of 40 real time projects is divided randomly in three parts before training is initiated: Training (70%), Validation (15%) and Testing (15%). Fig. 4 shows the data distribution.

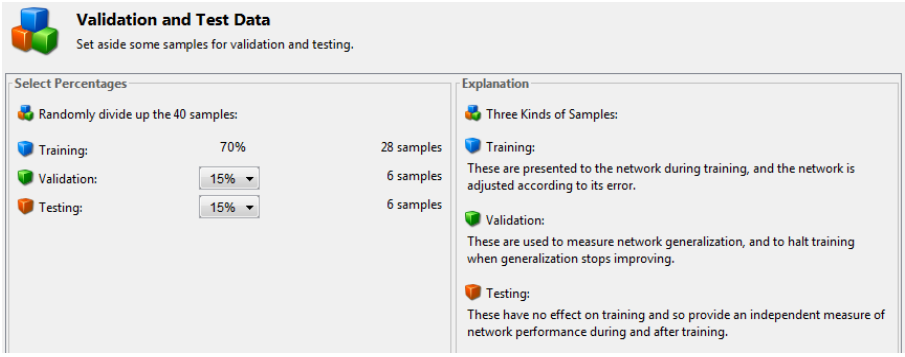


Fig. 4. Validation and test data percentage

Since the Levenberg Marquardt back-propagation optimization method is considered to be fast and uses less memory, this is being used for training the network. Neural network design configuration parameters are listed in Table 1.

Table 1. Network design configuration

S. no.	Parameters	Requirement phase	Construction phase	Testing phase
1	Training samples	40 samples of 4 elements (4 X 40) [Production effort, review effort, rework effort & prevention effort]		
2	Target samples	40 samples of 1 element (1 X 40) [Defect]		
3	Hidden layer	10	5	20
4	Data division for network use	Training (70%), validation (15%) & testing (15%)		

4.2 User interface for testing the framework with new projects

One of the primary objectives during GUI design has been to provide ease-of-use for project manager. The GUI based tool developed using Matlab R2013b uses only two windows, the first for identifying the phase for which prediction is required and the second to input the planned effort for activities and returns a straightforward output of the defect predictions (refer Fig. 5). For any new project, the project manager will provide the inputs required to the UI. The inputs would be the phase wise efforts planned for the project. Apart from production effort, the planned review effort, planned prevention effort and the planned rework effort are also required as a feed to the framework. Based on these inputs, the framework will forecast the number of defects that the project manager could expect to be discovered in various SDLC phases in the project. The defects are forecast in a range based manner. The framework would provide the minimum and maximum number of defects. The forecast would enable the project manager to plan prevention activities for the phase where the framework is projecting higher number of defects. The project manager can plan for multiple preventive actions, such as multiple review gates, usage of tools, and increasing review effort to mitigate the higher probability of defect leakage.

4.3 Results and discussion on NN based approach

In the experiments, the data sets with different network architectures have been used. The actual defects data from 40 completed projects was taken and used as a training data. Later, the framework was also tested for prediction of defects for newly started projects. The quality of fit and the predictive quality found for each of

the data sets have given very optimistic results (as discussed below). The prediction results indicate that the network (based on the proposed framework) tries to track the behavior of the full data set and its predicted value around the actual values –sometimes less and something more.

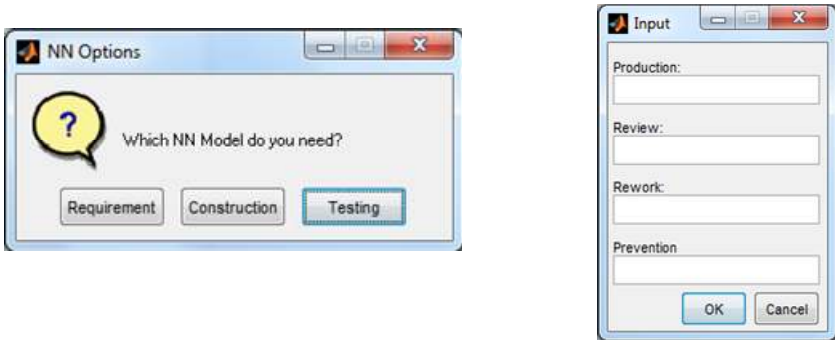


Fig. 5. Defect prediction system UI

For the test results of pilot conducted on 40 projects, software enhancement project dataset has correlation coefficient R value of 0.97 for Requirement gathering phase, 0.91 for construction phase and 0.89 for testing. The regression value depicts a closer relationship between the predicted and actual defects. The R value measures the correlation between outputs and targets. Fig. 6 represents the relationship for all three SDLC phases (Requirement gathering, Construction, Testing) for software enhancement project and defines the prediction reliability of the network designed.

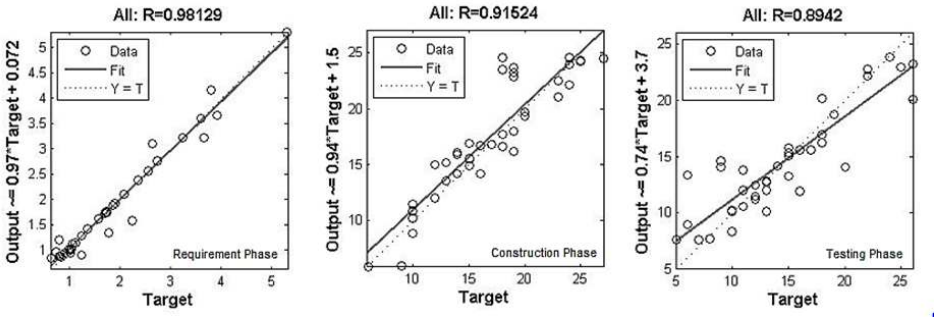


Fig. 6. Representation of R value for phases of software enhancement projects

Table 2 shows metrics which are often used by researchers to assess the performance of the model

- The mean absolute error (MAE) is used to measure how close forecasts or predictions are to the eventual outcomes.
- The mean squared error (MSE) is used to measure the average of the squares of the "errors", that is, the difference between the estimator and what is estimated.
- The root-mean-square error (RMSE) is used to measure differences between values (sample and population values) predicted by a model and the values actually observed [31].

As shown in Table 2, the MAE, MSE and RMSE values arrived from complete data set of 50 projects shows that ANN prediction results are in line with the Actual results. Also, the comparison of the defect trend chart (refer Fig. 7) from 10 projects during model validation phase shows that, in most cases, actual defects trend follows the ANN prediction trend.

Table 2. Comparative MAE, MSE and RMSE value for neural network model predictions and actual defects

Phase	Mean absolute error (MAE)	Mean squared error (MSE)	Root mean square error (RMSE)
	Actual defects vs NN predicted defects	Actual defects vs NN predicted defects	Actual defects vs NN predicted defects
Requirement	0.43	0.52	0.72
Construction	1.61	5.38	2.32
Testing	2.93	57.82	7.60

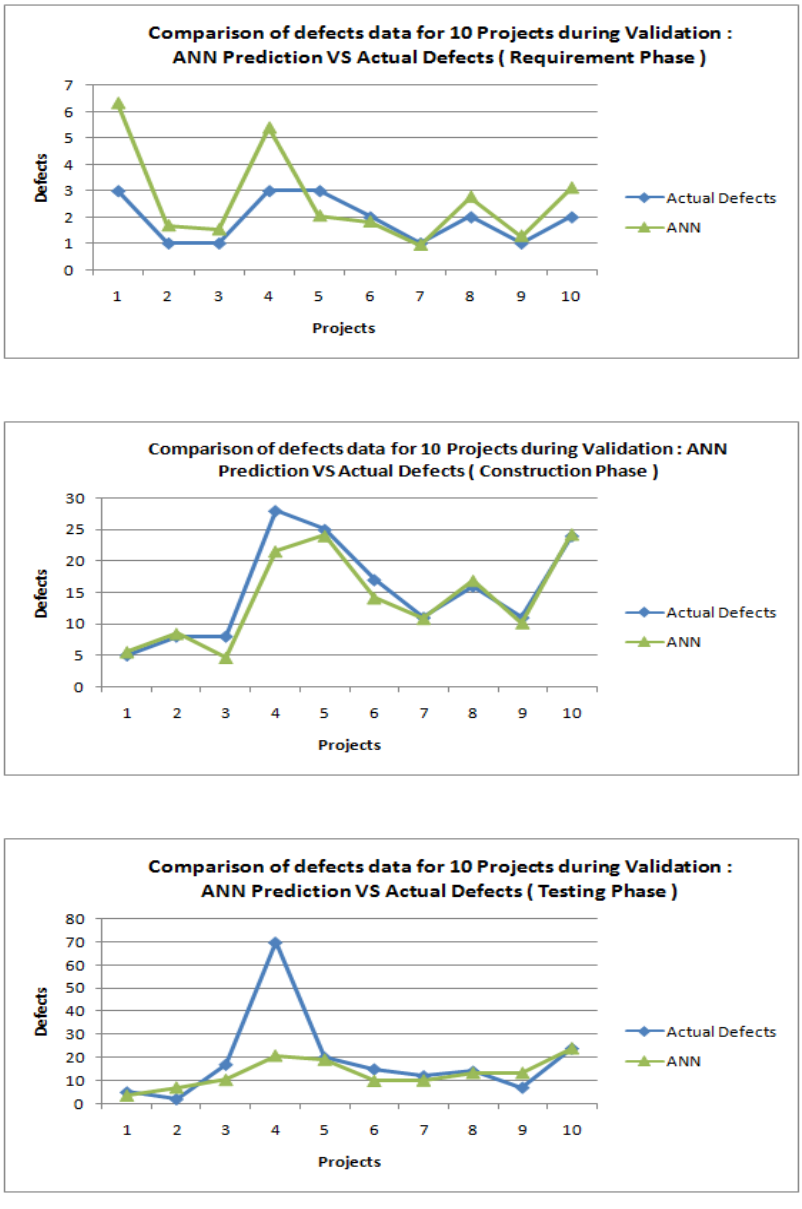


Fig. 7. Defect trend chart showing NN model prediction compared with actual defects

4.4 NN framework comparisons of results

The artificial neural network framework based approach appears to be promising for solving software engineering problems on defect prediction where historical data from past software enhancement projects is made available. The model performed well in the preliminary validation experiments.

The results from software enhancement project dataset has R value of 0.97 for Requirement gathering phase, 0.91 for construction phase and 0.89 for testing phase. The overall prediction reliability of the network designed is close to 92%. These results obtained by the proposed framework based on Neural Network are comparable with, and even better than, the results with accuracy of 88.09 % as obtained in [27], using public PROMISE library and Levenberg-Marquardt (LM) algorithm based neural network for predicting the software defects. The results are also better than with the work done on neural network based defect prediction model for Java projects following waterfall life cycle, having accuracy of 90% [5].

5 Conclusions

The results from experiments indicate that the proposed framework based on neural network approach possesses good properties from the standpoint of model quality of fit and predictive capability. The conclusions are based on investigations of software enhancement projects data set. In order to adapt the proposed framework to suit other software methodologies like ERP, Agile, Production Support, etc, further effort is required. The investigations in those respects will be reported in subsequent communications.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Levinson M. Let's stop wasting $78 billion per year. CIO Magazine; 2001.

[2] Shurei Tamura. Integrating CMMI and TSP/PSP: Using TSP data to create process performance models. Carnegie Mellon University; 2009.

[3] Vashisht V. Enhancing software process management through control charts. Journal of Software Engineering and Applications. 2014;7(2):87–93.

[4] Lanning DL, Khoshgoftaar TM. The impact of software enhancement on software reliability. IEEE Trans. Rel. 1995;44(4):677–682.

[5] Vashisht V, Lal M, Sureshchandar GS. A framework for software defect prediction using neural networks. Journal of Software Engineering and Applications. 2015;8(8):384–394.

[6] Pigoski Thomas M. Practical software maintenance: Best practices for managing your software investment. John Wiley & Sons, Inc.; 1996.

[7] Boetticher GD. Applying machine learners to GUI specifications in formulating early life cycle project estimations. Software Engineering with Computational Intelligence. 2003;1–16.

[8] Khoshgoftaar TM, Ed. Software engineering with computational intelligence; 2003.

[9] Sivanandam SN, Deepa SN. Principles of soft computing. John Wiley & Sons, Inc, 2nd Edition; 2009.

[10] Munakata T, Ed. Fundamentals of the new artificial intelligence. Texts in Computer Science; 2007.

[11] Narula SC, Wellington JF. Prediction, linear regression and the minimum sum of relative errors. Technometrics. 1977;19:185- 190.

[12] Gafhey JE Jr. Estimating the number of faults in code. IEEE Transactions on Software Engineering. 1984;SE10:459-464.

[13] Karunanithi N, Malaiya YK, Whitley D. Prediction of software reliability using neural networks. Proceedings of the International Symposium on Software Reliability Engineering. 1991;124-130.

[14] Kumar S, Krishna BA, Satsangi PJ. Fuzzy systems and neural networks in software engineering project management. Journal of Applied Intelligence. 1994;4:31-52.

[15] Srinivasan K, Fisher D. Machine learning approaches to estimating software development effort. IEEE Trans. Software Engineering. 1995;126-137.

[16] Boetticher G, Srinivas K, Eichmann D. A neural net-based approach to software metrics. Proceedings of the 5th International Conference on Software Engineering and Knowledge Engineering. 1993; 271-274.
Available: http://nas.cl.uh.edu/boetticher/publications.html

[17] Boetticher G, Eichmann D. A neural net paradigm for characterizing reusable software. Proceedings of the First Australian Conf on Software Metrics. 1993;41-49.
Available: http://nas.cl.uh.edu/boetticher/publications.html

[18] Boetticher G. Characterizing object-oriented software for reusability in a commercial environment. Reuse '95 Making Reuse Happen - Factors for Success, Morgantown, WV; 1995.
Available: http://nas.cl.uh.edu/boetticher/pub lications.html

[19] Boetticher G. An assessment of metric contribution in the construction of a neural network-based effort estimator. Second lnt. Workshop on Soft Computing Applied to Soft. Engineering; 2001.
Available: http://nas.cl. uh.edu/boetticher/publications.html

[20] Boetticher G. Using machine learning to predict project effort: empirical case studies in data-starved domains. Workshop on Model-Based Requirements Engineering; 2001.
Available: http://nas.cl.uh.edu/boetticher/publications.html

[21] McCabe TJ. A complexity measure. IIEEE Trans. Software Eng. 1976;SE-2(4):308–320.

[22] Curtis B, Sheppard SB, Milliman P, Borst MA, Love T. Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. IEEE Trans. Software Eng. 1979;SE-5:96 -104.

[23] Gayathri M, Sudha A. Software defect prediction system using multilayer perceptron neural network with data mining. International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878. 2014;3(2):54-59.

[24] Crosby P. Quality is free: The art of making quality certain. McGraw-Hill, New York; 1979.

[25] Singh S, Singh M. Software defect prediction using adaptive neural networks. International Journal of Applied Information Systems. 2012;4(1):29–33.

[26] Katiyar N, Singh R. Prediction of software development faults using neural network. VSRD-IJCSIT. 2011;1(8):556-566.

[27] Singh M, Singh Salaria D. Software defect prediction tool based on neural network. IJCA. 2013;70(22):22–28.

[28] Juran J, Gryna F. Quality control handbook. 4th ed., McGraw-Hill, New York; 1988.

[29] Sunohara Takeshi, et al. Program complexity measure for software development management. Proceedings of the 5th International Conference on Software Engineering. IEEE Press; 1981.

[30] Neural Network Toolbox™ User's Guide; 2015.

[31] Chai T, Draxler RR. Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature. Geosci. Model Dev. 2014;7(3):1247–1250.

Peer-review history:
The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)
http://sciencedomain.org/review-history/14542