



Defect prediction model of static code features for cross-company and cross-project software

Satwinder Singh¹ · Rozy Singla²

Received: 17 July 2017 / Accepted: 26 November 2018 / Published online: 6 December 2018
© Bharati Vidyapeeth's Institute of Computer Applications and Management 2018

Abstract Software project metrics are seen needless in software industries but they are useful when some unacceptable situations come in the project (Satapathy et al., Proceedings of the 48th annual convention of CSI, vol 2, 2013). Mainly the focus of various defect prediction studies is to build prediction models using the regional data available within the company. So companies maintain a data repository where data of their past projects can be stored which can be used for defect prediction in the future. However, many companies do not follow this practice. In software engineering, the crucial task is Defect prediction. In this paper, a binary defect prediction model was built and examined if there is any conclusion or not. This paper presents the assets of cross-company and within-company data against software defect prediction. Neural network approach has been used to prepare the model for defect prediction. Further, this paper compares the results of within and cross-company defect prediction models. To analyse the results for with-in company two versions of Firefox (i.e. 2.0 and 3.0) were considered; for cross project one version of Mozilla Sea Monkey (1.0.1); for cross-company validation one version of LICQ were considered. Main focus of the study is to analyse the behavior or role of software metrics for acceptable level of defect prediction.

Keywords Defect prediction · Object-oriented metrics · Artificial neural network (ANN) · Cross company defect prediction (CCDP)

1 Introduction

Mostly the testing is done during the development phase. There is no post maintenance. It can help to detect future defects in the system as well as it can help to build a model for defect prediction [1].

Most of the today's organizations still searching for a defect prediction model which can be used for any type of software but still it is under the development. Generally the prediction models focus on following aspects:

1. Finding the bugs in software system;
2. Checking the reliability of the software against the time frame;
3. To grasp the effect of designing process over defects and failures.

The most famous and widely used technique for defect prediction is testing. Testing efforts depend on the size of the project. Testing maybe simple or complicated depended on the project size. Defects can be easily predicted in other projects or other parts of project if any relation can be established between software metrics and defects [2].

Object-oriented patterns are widely used in software development. Software metrics are used as quality predictor for OO software. Various researchers and practitioners suggested various metrics to calculate the quality of the software [3].

Software metrics are collected with the help of automated tools which are used by defect prediction models to predict the defects in the system. There is generally a

✉ Satwinder Singh
satwinder.singh@cup.edu.in

Rozy Singla
rozysingla92@gmail.com

¹ Department of Computer Science and Technology, Central University of Punjab, Bathinda 151001, India

² MOM Department, GWPC, Sanganer, Jaipur 302033, India

dependent variable and various independent variables are present in any fault prediction model. Dependent variable defines that the software modules are faulty or not. Various metrics such as process metrics, product metrics etc. can be used as independent variables. For example cyclomatic complexity and lines of code which are method-level product metrics [4].

Cross company defect prediction (CCDP) is a mechanism that builds defect predictors by using data from various other softwares and companies and the data may be heterogeneous in nature. Cross project defect prediction use data from with-in company projects or cross-company projects. Cross-company (CC) data involves facts from many different projects and are diversified as compared to with-in company (WC) data [5]. To solve this heterogeneity number of analysis is required and it is been done by various research community members. Having a generalized prediction model for defect prediction will help the maintenance and testing team to analyse the software at the best. This study focus on to use the software metrics to develop the prediction model for defects prediction. Prediction models proposed in this study include prediction models for within-company defect prediction, cross-project defect prediction and cross-company defect prediction. Further, efforts has been made to check and compare the accuracy of each with various mathematical parameters.

2 Literature survey

Zimmermann et al. [6] calculated the performance of defect prediction for cross projects by using data from 12 projects (622 combinations). Among of these combinations, only 21 pairs resulted in efficient prediction performance. Data distributions of the initial and final projects are different which results in low prediction performance. It is expected that training and test data have the same distribution of data. This assumption is good for within-project prediction and may be not suited for cross-project prediction. Cross-project prediction can be indicted in two dimensions: the domain and the company. Zimmermann et al. noticed that in many software companies may or may not provide local data for defect prediction as they are small or they do not have any past data. Zimmermann et al. observed the data sets from F. and IE. They experimented on these web browsers and found that F. data could predict the defects in IE very well, but vice versa was not true. They come up with the result that “building a model from a small population to predict a larger one is likely more difficult than the reverse direction”.

Zhang et al. [7] proposed a universal model for defect prediction that can be used in with-in company and cross company projects. One issue in building a cross company

defect prediction is the variations in data distribution. To overcome this, the authors first suggested collecting data, and then transforming the training and testing data to make more similar in their data distribution. They proposed a rank transformation which is context-free to limit the changes in the distribution of data before applying them to the universal defect prediction model. The six context factors is used by authors for prediction. They used 21 code metrics and 5 process metrics in their research. Their experiment results shows higher AUC values and higher recall than with-in project models and has better AUC for cross-projects.

Ma et al. [8] proposed a Transfer Naive Bayes (TNB) algorithm for defect prediction in cross-company projects, which is a novel transfer learning algorithm. The advantage of transfer learning is that it allows that training and testing data to be heterogeneous. They have used instance-transfer approach in their research which assigns weights to source instances according to their contribution in the prediction model. They use four performance metrics, probability of detection (PD), F-measure, probability of false alarms (PF), and AUC to measure the performance of defect predictor. They show that the TNB gives good performance.

Mahaweerawat et al. [9] introduced a new approach in object-oriented software systems for predicting faults. In this neural network is used with supervised learning. They used multi-layer perceptron (MLP) neural network with back-propagation to identify fault-prone classes and radial basis function (RBF) neural network is used to cluster the faults of same types. Their experiment results show 90% accuracy for predicting faultiness of a module/class.

Aggarwal et al. [10] proposed a model to discover the dependency of faults on object-oriented design metrics of a software product. They used data from Java applications which contains 136 classes. They used Principal component method for preprocessing of data. Univariate Logistic Regression is used to check the effect of software metrics on fault proneness. Prediction model is developed using Multivariate Logistic Regression. The model gives sensitivity 86.5% and specificity above 90%.

Various Software metrics are used by Singh and Salaria [11] to find the Software defects. They used various machine leaning methods for defect prediction. They discussed about the uses of neural network in various fields such as data mining, image processing, etc. Experiment Data is collected from PROMISE repository. The data is divided in the ratio of 17:3 for training and testing. Levenberg–Marquardt (LM) algorithm is used for training which results in 88% accuracy.

Canfora et al. [12] proposed a multi-objective approach for cross-company defect prediction, which uses logistic regression model, developed using a genetic algorithm. Multivariate logistic regression is used in this experiment. It deals with the defect prediction, and the cost-

Table 1 Dataset details

Name	Number of classes	Number of defect classes	Defects (%)
Firefox Version 2.0	4524	81	1.79
Firefox Version 3.0	4971	59	1.186
SM 1.0.1	4103	47	1.145
Licq	280	126	45

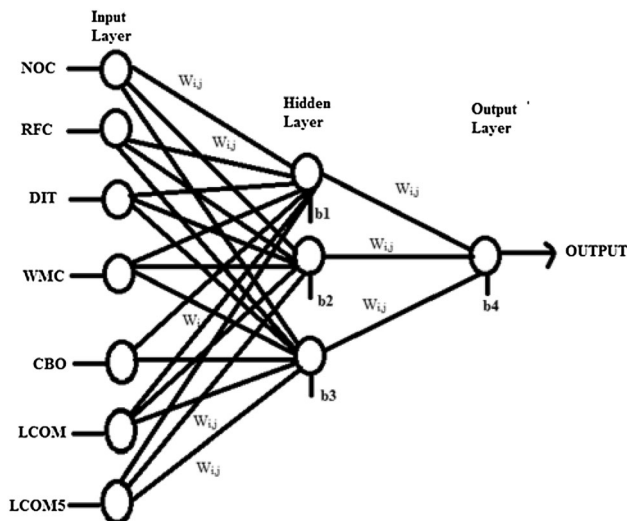


Fig. 1 Feed forward neural network

effectiveness. They used a multi-objective Genetic Algorithm (GA) is used for training, in which metrics are used as independent variables. They used ten datasets from the Promise repository. They perform a data standardization to reduce the effect of heterogeneity. The model gets a better cost-effectiveness than within-project predictors, and gives better results than single-objective predictors.

Lessmann et al. [13] use metric-based classification for defect prediction. They designed a defect prediction model which uses data from 10 public-domain data sets collected from the NASA Metrics Data (MDP) repository and the PROMISE repository and 22 classification methods are tested against defect prediction. They used area under the receiver operating characteristics curve (AUC) for measuring the performance of the model. They used various classifiers which are divided into several categories such as neural networks, statistical approaches, support vector machines, ensembles, nearest-neighbor methods and tree-based methods. They divide the data randomly as 2/3 for training and 1/3 for performance evaluation. Results show more than 0.7 AUC for most of the classifiers. They observed that RndFor, LS-SVMs, MLPs, and Bayesian networks which are sophisticated classifiers produce the better results. Simple classifiers are good enough to analyze the correlation between static code attributes and software defects.

Kumar et al. [14] compare fuzzy logic and artificial neural network methods for predicting the defect density

(DD) of software. They used mean absolute error (MAE), root mean square error (RMSE) and graphical analysis for performance measurement. Defect density (DD) is an attribute used to the reliability of the software product. They used data from two projects of different domains. Fuzzy inference system (FIS) gives maximum 77% and minimum 73% accuracy. ANN gives up to 85% accuracy and 0.3872 as RMSE.

Kaur et al. [15] presented a survey on various object-oriented metrics as quality factors for software. They used 22 software metrics for their research. They used data from 3 projects. They identified the metrics which can be used to check the quality level of the software.

Kaur et al. [16] designed a framework to identify software code smells to analyze the quality of the software. They used feed forward neural network (FFNN) and used eight object-oriented metrics for their research. Their framework provides a better result and they also show a relationship between object-oriented metrics and bad smells.

3 Collection of data

Data is collected from the Bugzilla database for two versions of Mozilla Firefox 2.0 and 3.0 and for one version of Mozilla Seamonkey. The Bugzilla database consists of all errors (bugs) that have been found in projects with detailed information. Another system chosen for cross company and cross domain analysis is Licq (UNIX based). The database for bugs is obtained from social community known as GitHub¹ community. Licq has 280 classes only and is smallest among all dataset (Table 1).

4 Multi layer preceptron model (MLP) based on neural network

Various techniques are adopted for defect prediction such as Naïve Byes, Random Forest, SVM, Machine Learning etc. In this paper FFNN is proposed.

FFNN uses a back-propagation learning algorithm. It is used to solve a vast variety of problems. In this various neurons are organized into various different layers such as

¹ <http://www.github.com>.

Table 2 Confusion matrix for defect prediction

	Non-faulty	Faulty
Non-faulty	True negative (TN)	False positive (FP)
Faulty	False negative (FN)	True positive (TP)

Table 3 Results for 500 epochs

Training on	Testing on	%Accuracy	MSE	AUC	Precision
Ver. 2.0	Ver. 2.0	98.19	0.0181	0.707	0.4286
	Ver. 3.0	98.77	0.0123	0.293	0.375
	SM 1.0.1	98.73	0.0219	0.303	0.2727
	Licq	55.36	0.0446	0.590	1.000
Ver. 3.0	Ver. 2.0	98.19	0.0181	0.439	0.4286
	Ver. 3.0	98.77	0.0123	0.752	0.375
	SM 1.0.1	98.76	0.0217	0.364	0.300
	Licq	55.36	0.04464	0.349	1.000
SM 1.0.1	Ver. 2.0	97.90	0.0245	0.346	0.150
	Ver. 3.0	98.47	0.0243	0.268	0.160
	SM 1.0.1	98.32	0.0329	0.817	0.1333
	Licq	56.07	0.04536	0.503	1.000
Licq	Ver. 2.0	19.39	0.3763	0.685	0.0207
	Ver. 3.0	20.18	0.3753	0.699	0.0139
	SM 1.0.1	21.79	0.3738	0.734	0.0141
	Licq	57.14	0.2434	0.600	0.5517

input layer, output layer, and hidden layers. Figure 1 show the model used in this paper. Different layer are connected to each other.

Weights are assigned against each connection between two neurons (i,j) the weight coefficient w_{ij} . It has an impact on the importance of the connection in the FFNN. Following equation can be used to determine the output of a layer

$$a = x_1W_1 + x_2W_2 + x_3W_3 \dots + x_nW_n. \tag{1}$$

In this paper input layer consists of seven neurons and hidden layer contains three neurons. The inputs neurons used are object-oriented metrics which includes:-NOC [3],

RFC [3], DIT [3], WMC [3], CBO [3], LCOM [3], LCOM5 [17, 18].

Various artificial neural network (ANN) experiments use multi-layer perceptron (MLP) method. MLP is a FFNN that uses back propagation algorithm as its training algorithm. A general framework is provided by FFNN for indicating mappings between input variables and output variables. For this various activation functions are used which represents the nonlinear function of various variables in terms of formations of nonlinear functions of a single variable.

In this paper, Hyperbolic Tangent Sigmoid Function (tansig) and Linear Transfer Function (purelin) are used as activation functions for the layers.

5 Performance evaluation parameters

Performance evaluation parameters are needed to validate the performance of the proposed model. In this paper parameters required to validate the performance were calculated using the confusion matrix. If these parameters are not considered then the performance of the proposed model can't be evaluated. In this section, various parameters are defined which were used to calculate the performance of the proposed model.

Table 2 shows the confusion matrix for fault prediction. There are four categories of confusion matrixes which are as following:

1. True positives (TP): number of classes which are classified as faulty classes correctly.
2. False positives (FP): number of not-faulty classes predicted as faulty classes.
3. True negatives (TN): number of non-faulty modules correctly predicted as non-faulty.
4. Finally, false negatives (FN): number of faulty classes incorrectly predicted as not-faulty classes [19].

Performance parameter used to measures the proposed model's performance:

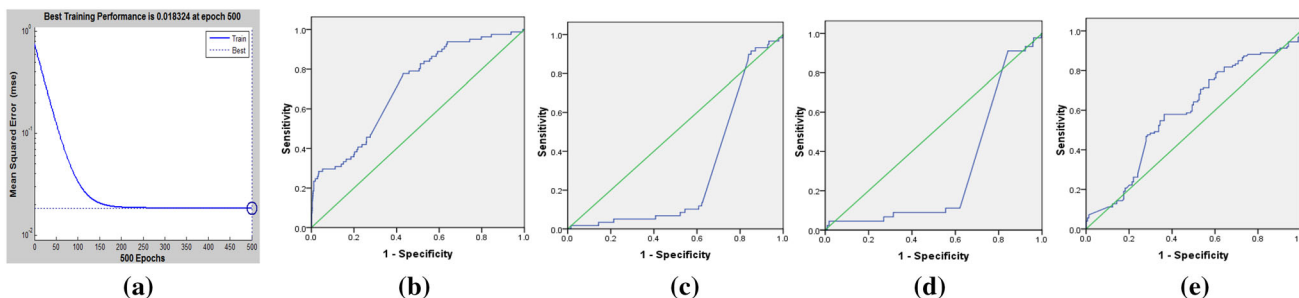


Fig. 2 a Training performance of Ver. 2.0, ROC curve for b Ver. 2.0 using Ver. 2.0, c Ver. 3.0 using Ver. 2.0, d SM 1.0.1 using Ver. 2.0, e Licq using Ver. 2.0

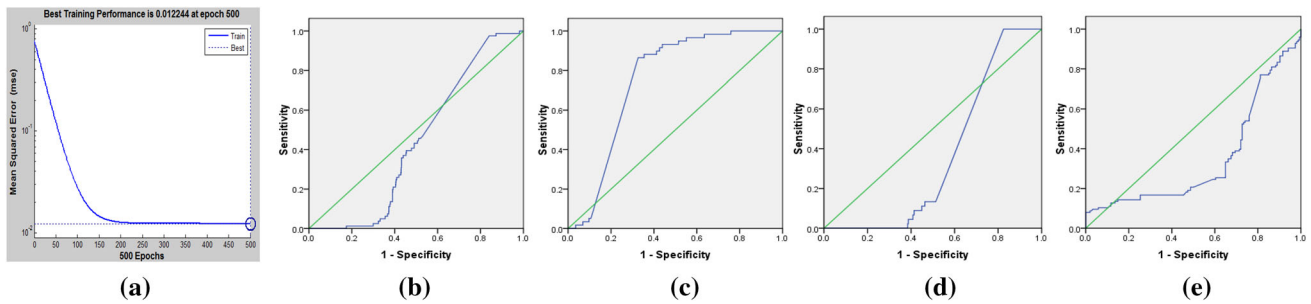


Fig. 3 a Training performance of Ver. 3.0, ROC curve for b Ver. 2.0 using Ver. 3.0, c Ver. 3.0 using Ver. 3.0, d SM 1.0.1 using Ver. 3.0, e Licq using Ver. 3.0

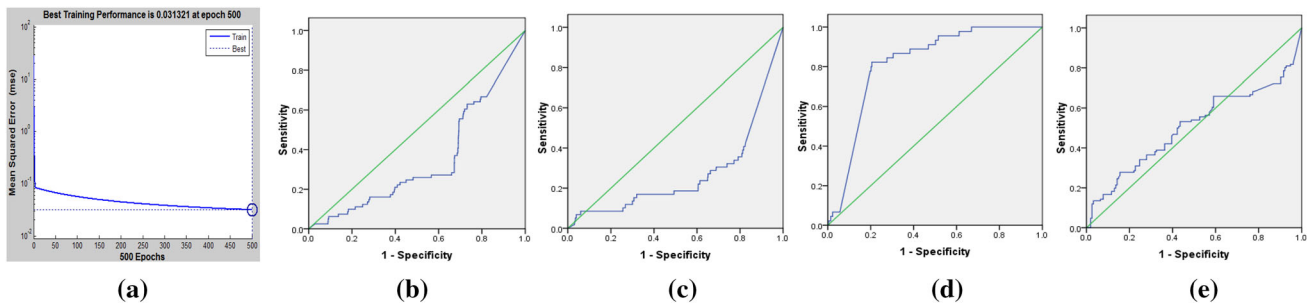


Fig. 4 a Training performance of SM 1.0.1, ROC curve for b Ver. 2.0 using SM 1.0.1, c Ver. 3.0 using SM 1.0.1, d SM 1.0.1 using SM 1.0.1, e Licq using SM 1.0.1

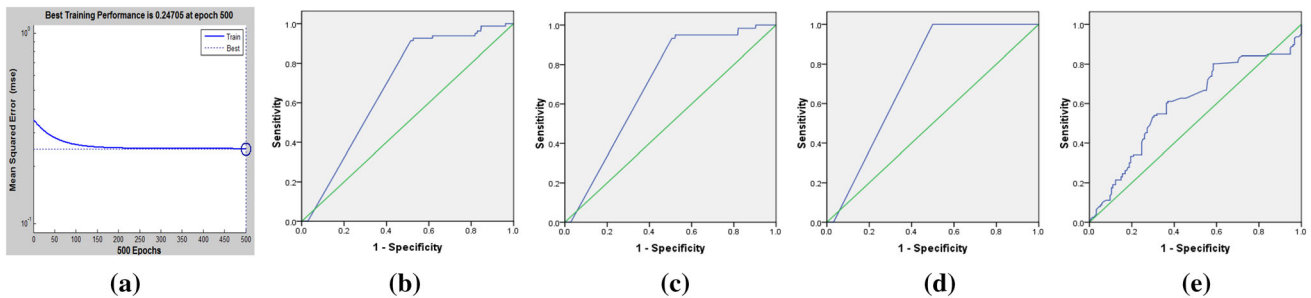


Fig. 5 a Training performance of Licq, ROC curve for b Ver. 2.0 using Licq, c Ver. 3.0 using Licq, d SM 1.0.1 using Licq, e Licq using Licq

Precision

It shows that how many measurements produce same results when conditions are unchanged. Precision refers to the closeness of two or more measurements to each other.

$$\text{Precision} = \frac{TP}{FP + TP} \tag{2}$$

Accuracy

It is ratio of correctly classified modules and all modules. It is defined as:

$$\text{Accuracy} = \frac{TN + TP}{TP + TN + FP + FN} \tag{3}$$

Mean square error (MSE)

The MSE measures the squares of the “errors” and uses average value, i.e. the difference between the actual and predicted values.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y'_i - y_i)^2 \tag{4}$$

where n = total number of samples, y is the output generated by the model and y' is the expected output.

Receiver operating characteristics (ROC)

The performance of a binary classifier is shown by ROC curve. It is in a graphical form. The true positive rate (TPR) and the false positive rate (FPR) are used to plot the curve. The Area Under ROC Curve (AUC) is used to analyze different ROC curves. Higher AUC values indicate the classifier is good [20].

Table 4 Results for 1000 Epochs

Training on	Testing on	%Accuracy	MSE	AUC	Precision
Ver. 2.0	Ver. 2.0	98.21	0.0182	0.730	0.500
	Ver. 3.0	98.77	0.0123	0.743	0.375
	SM 1.0.1	98.76	0.0217	0.767	0.300
	Licq	55.36	0.04464	0.400	1.000
Ver. 3.0	Ver. 2.0	98.21	0.0179	0.654	0.500
	Ver. 3.0	98.77	0.0123	0.821	0.375
	SM 1.0.1	98.76	0.0217	0.815	0.300
	Licq	55.36	0.04464	0.351	1.000
SM 1.0.1	Ver. 2.0	98.12	0.0199	0.621	0.300
	Ver. 3.0	98.65	0.0183	0.690	0.250
	SM 1.0.1	98.59	0.0258	0.813	0.2105
	Licq	56.07	0.04429	0.645	1.000
Licq	Ver. 2.0	76.55	0.1958	0.320	0.0040
	Ver. 3.0	77.59	0.1929	0.233	0.000
	SM 1.0.1	76.55	0.2044	0.235	0.0011
	Licq	57.14	0.2492	0.461	0.875

6 Results

In this paper Accuracy, Precision, MSE, and ROC curve are used to evaluate the performance of the model. More accuracy means the model performs better.

Table 3 shows results for 500 epochs where model is trained on one dataset and is tested on all datasets.

From the above results it was analyzed that the Licq dataset has highest precision i.e. 1 means Licq produces the outputs which are more closed to each other. But the Licq doesn't give accuracy. The Ver. 3.0 has lowest precision i.e. 0.0139 but it has highest accuracy for defect prediction i.e. 98.77% when tested using Ver. 2.0 and Ver. 3.0. The accuracy of proposed model is up to 99% which means the proposed model with 500 epochs has good accuracy rate while applying it over the same version or subsequent versions. Corresponding graphs of above data is shown in Figs. 2, 3, 4 and 5.

Table 4 shows the results of proposed model for 1000 epochs. In this the dataset is trained using 1000 iterations.

From the above results it was examined that the proposed model gives accuracy in the range from 55 to 99%. The highest accuracy is shown by Ver. 2.0 and 3.0. The accuracy has higher values indicates that the model proposed in this paper effectively identifies defective modules in the software. Corresponding graphs of above data are shown in Figs. 6, 7, 8 and 9.

In the Table 5, the results of proposed model are shown using 2000 epochs i.e. the training is performed using 2000 iterations.

Using the Table 5 it is examined that the models have MSE values below 0.4. Ver. 3.0 has MSE value 0.0123

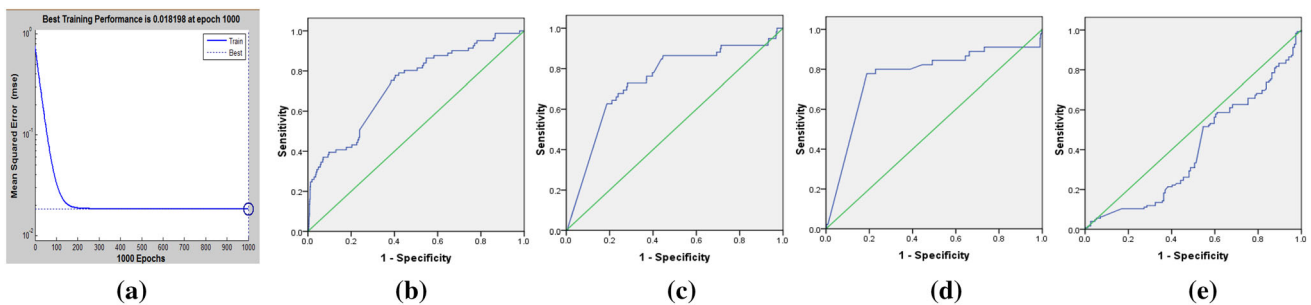


Fig. 6 a Training performance of Ver. 2.0, ROC curve for b Ver. 2.0 using Ver. 2.0, c Ver. 3.0 using Ver. 2.0, d SM 1.0.1 using Ver. 2.0, e Licq using Ver. 2.0

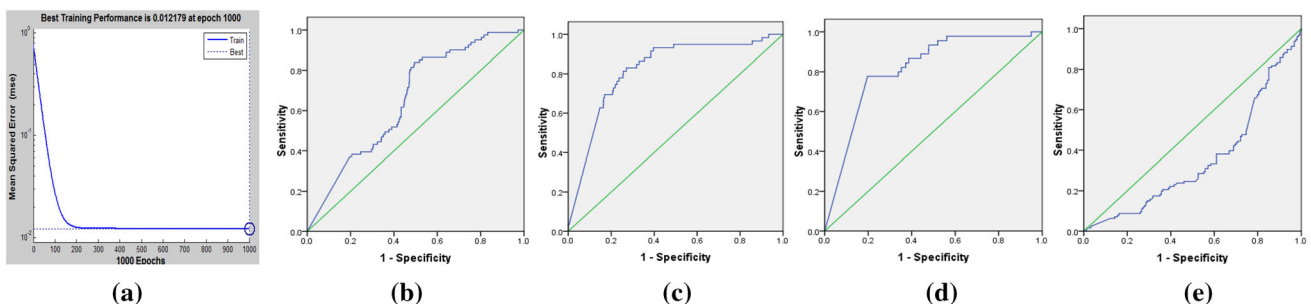


Fig. 7 a Training performance of Ver. 3.0, ROC curve for b Ver. 2.0 using Ver. 3.0, c Ver. 3.0 using Ver. 3.0, d SM 1.0.1 using Ver. 3.0, e Licq using Ver. 3.0

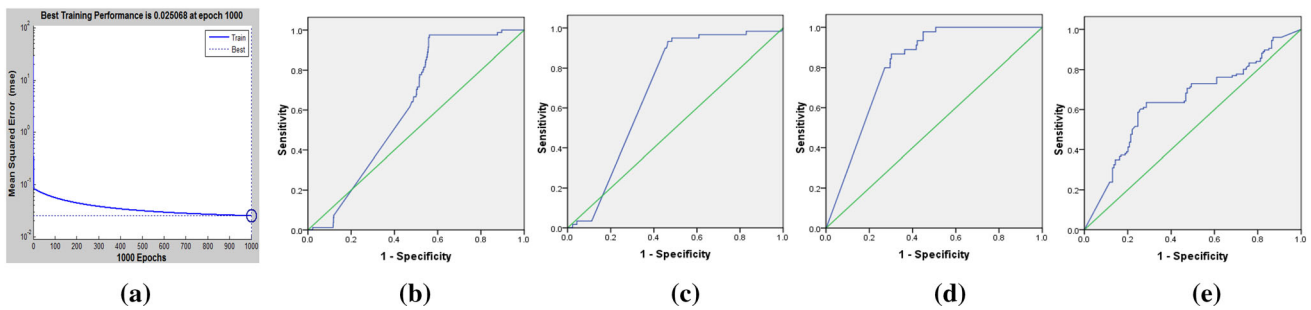


Fig. 8 a Training performance of SM 1.0.1, ROC curve for **b** Ver. 2.0 using SM 1.0.1, **c** Ver. 3.0 using SM 1.0.1, **d** SM 1.0.1 using SM 1.0.1, **e** Licq using SM 1.0.1

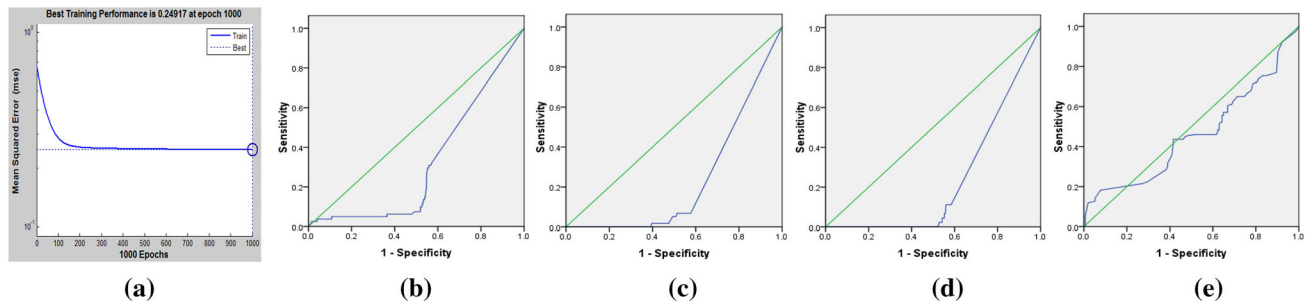


Fig. 9 a Training performance of Licq, ROC curve for **b** Ver. 2.0 using Licq, **c** Ver. 3.0 using Licq, **d** SM 1.0.1 using Licq, **e** Licq using Licq

Table 5 Results for 2000 Epochs

Training on	Testing on	%Accuracy	MSE	AUC	Precision
Ver. 2.0	Ver. 2.0	98.23	0.0177	0.681	0.600
	Ver. 3.0	98.77	0.0123	0.770	0.375
	SM 1.0.1	98.78	0.0214	0.764	0.3333
	Licq	55.36	0.04464	0.669	1.000
Ver. 3.0	Ver. 2.0	98.21	0.0179	0.700	0.500
	Ver. 3.0	98.77	0.0123	0.705	0.375
	SM 1.0.1	98.78	0.0214	0.811	0.333
	Licq	55.36	0.04464	0.602	1.000
SM 1.0.1	Ver. 2.0	98.19	0.0181	0.654	0.4286
	Ver. 3.0	98.75	0.0137	0.787	0.333
	SM 1.0.1	98.73	0.0219	0.700	0.2727
	Licq	55.36	0.04464	0.652	1.000
Licq	Ver. 2.0	38.26	0.2933	0.709	0.0265
	Ver. 3.0	36.41	0.2978	0.726	0.0177
	SM 1.0.1	36.66	0.3010	0.772	0.0174
	Licq	57.50	0.2433	0.629	0.5574

using Ver. 2.0 and Ver. 3.0 which is lowest among all. The model performs better with cross projects, but doesn't show accuracy in defect prediction using cross-company projects The Licq dataset has highest precision i.e. 1 when tested using same version and the SM 1.0.1 has lowest

precision i.e. 0.0174 when tested using Licq. Related graphs of above table are shown in Figs. 10, 11, 12 and 13.

7 Conclusion

Results show that if more training is used, better results can be produced as with 500 epochs, model gives better results only for with-in company projects but with 1000 and 2000 epochs it works good for cross projects also as shown in Tables 4 and 5. But in case of Licq dataset the results of the model are not improved. The reason behind this may be the compact size of the Licq dataset.

As compare to previous work, these results are better. The proposed model gives AUC value 0.821 using Firefox Ver. 3.0 on Firefox Ver. 3.0, 0.815 for SM 1.0.1 when the model is trained with Firefox Ver. 2.0. The model proposed by [7] with the use of clustering as classification and Cliff ranking as analysis, is tested on few datasets, so it may not applicable for other datasets. Further, they had boldly written it as limitation of analysis. The model proposed by [8] with the help of Naïve Bayes technique also helps to transfer the results of one dataset to others to predict defects in the dataset. It doesn't provide any defined model for cross-project and cross-company projects. The model propose by [21] with the Random Forest analysis technique uses a Just In Time (JIT) for defect prediction, which requires more training for more accurate results. After

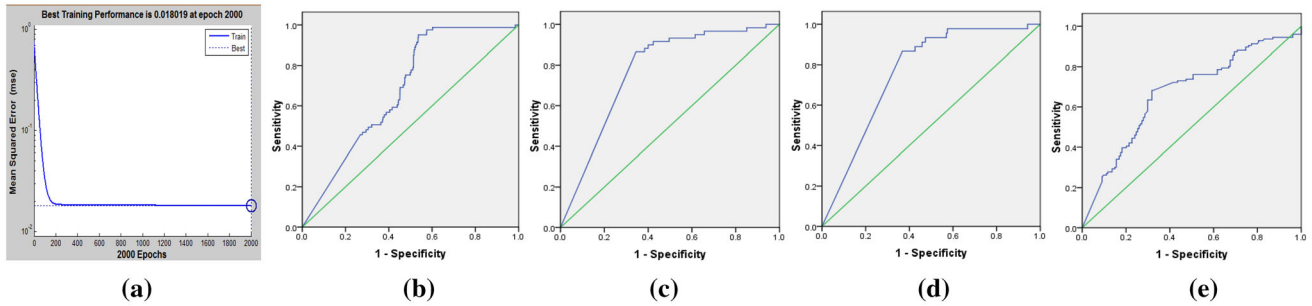


Fig. 10 a Training performance of Ver. 2.0, ROC curve for b Ver. 2.0 using Ver. 2.0, c Ver. 3.0 using Ver. 2.0, d SM 1.0.1 using Ver. 2.0, e Licq using Ver. 2.0

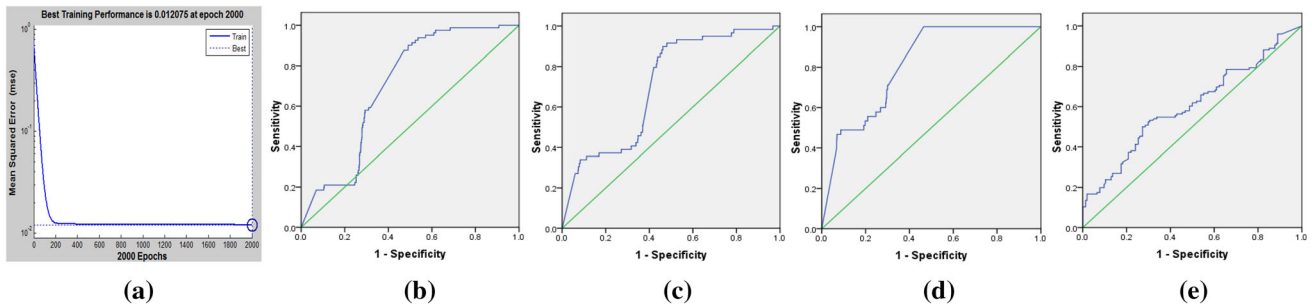


Fig. 11 a Training performance of Ver. 3.0, ROC curve for b Ver. 2.0 using Ver. 3.0, c Ver. 3.0 using Ver. 3.0, d SM 1.0.1 using Ver. 3.0, e Licq using Ver. 3.0

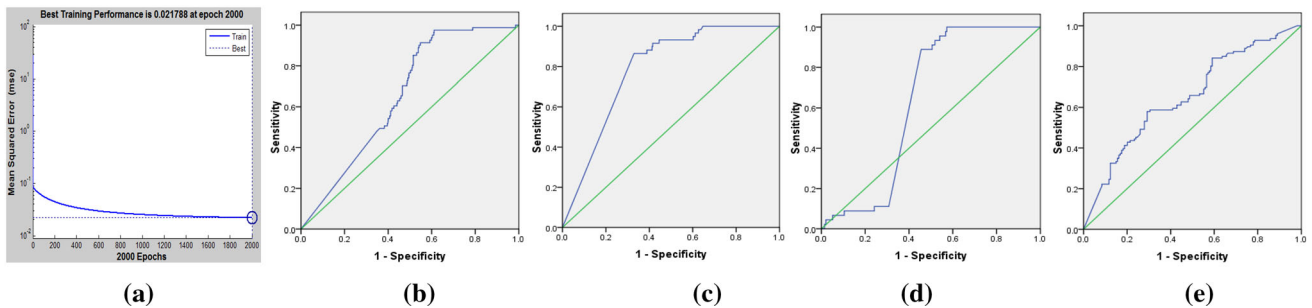


Fig. 12 a Training performance of SM 1.0.1, ROC curve for b Ver. 2.0 using SM 1.0.1, c Ver. 3.0 using SM 1.0.1, d SM 1.0.1 using SM 1.0.1, e Licq using SM 1.0.1

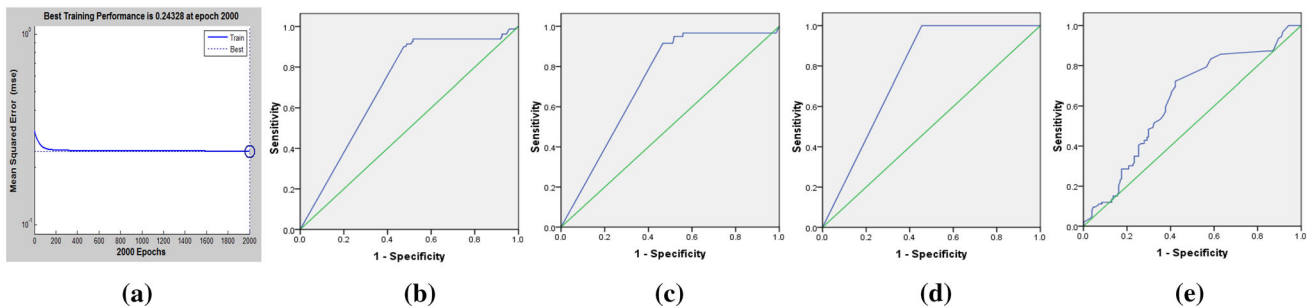


Fig. 13 a Training performance of Licq, ROC curve for b Ver. 2.0 using Licq, c Ver. 3.0 using Licq, d SM 1.0.1 using Licq, e Licq using Licq

analyzing these results, we can say that proposed model is well suited for predicting defects in both with-in company projects as well as in cross projects but for cross-company

projects, results are not as good enough as compare to with-in and cross-project models. The reasons for it may lie under various domains and require to be analysed to

Table 6 Comparison to previous work

	Proposed model	Jang et al. [7]	Ma et al. [8]	Fukushima et al. [21]
Technique used	Multi layer perceptron NN model	Clustering and cliffs ranking	Naïve Bayes	Random forest
AUC for with-in company Projects	0.821	0.80	0.77	0.81
AUC cross-projects	0.815	0.79	N/A	0.79
AUC cross-company projects	0.722	0.82	0.71	N/A

achieve more accuracy. To get more accuracy of the prediction model various other techniques of machine learning can be tested (Table 6).

References

- Mittal P, Singh S, Kahlon KS (2011) Identification of error prone classes for fault prediction using object oriented metrics. In: Abraham A et al (eds) ACC 2011, Part II, CCIS 191, pp 58–68
- Rawat MS, Dubey SK (2012) Software defect prediction models for quality improvement: a literature study. *IJCSI Int J Comput Sci* 9(5):288–296
- Chidamber SR, Kemerer CF (1994) A metrics suite for object oriented design. *IEEE Trans Softw Eng* 20(6):476–493
- Singh S, Kahlon KS, Sandhu PS (2010) Re-engineering to analyze and measure object oriented paradigms. In: 2010 2nd IEEE international conference on information management and engineering, ICIME
- Kitchenham BA, Mendes E (2004) A comparison of cross-company and within-company effort estimation models for web applications. In: *Proceedings Metrics'04*. IEEE Computer Society, Chicago
- Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B (2009) Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: *Proceedings of ESEC/FSE 2009*
- Zhang F, Mockus A, Keivanloo I, Zou Y (2014) Towards building a universal defect prediction model. In: *MSR 2014 proceedings of 11th working conference on mining software repositories*, pp 182–191
- Ma Y, Luo G, Zang X, Chen A (2012) Transfer learning for cross-company software defect prediction. *Inf Softw Technol* 54:248–256
- Mahaweerawat A, Sophatsathit P, Lursinsap C, Musilek P (2015) Fault prediction in object-oriented software using neural network techniques. In: *Proceedings of the InTech conference*, Huston, pp 27–34
- Aggarwal KK, Singh Y, Kaur A, Malhotra R (2007) Investigating effect of design metrics on fault proneness in object-oriented systems. *J Object Technol* 6(10):127–141
- Singh M, Salaria DS (2013) Software defect prediction tool based on neural network. *Int J Comput Appl* 70(22):22–28
- Canfora G, Lucia AD, Penta MD, Oliveto R, Panichella A, Panichella S (2013) Multi-objective cross-project defect prediction. In: *Proceedings of the 6th IEEE international conference on software testing, verification and validation*. IEEE, Luxembourg, pp 252–261
- Lessmann S, Baesens B, Mues C, Pietsch S (2008) Benchmarking classification models for software defect prediction: a proposed framework and novel findings. *IEEE Trans Softw Eng* 34(4):485–496
- Kumar Vijai, Sharma Arun, Kumar Rajesh (2013) Applying soft computing approaches to predict defect density in software product releases: an empirical study. *Comput Inform* 32:203–224
- Kaur A, Singh S, Kahlon KS (2009) A metric framework for analysis of quality of object oriented design. *Int J Comput Inf Eng* 3(12):2875–2878
- Kaur J, Singh S (2016) Neural network based refactoring area identification in software system with object oriented metrics. *Indian J Sci Technol* 9(10)
- Hitz M, Montazeri B (1996) Chidamber and Kemerer's metrics suite: a measurement theory perspective. *IEEE Trans Softw Eng* 22(4):267
- Singh S, Kaur S (2017) A systematic literature review: Refactoring for disclosing code smells in object oriented software. *Ain Shams Eng J*
- Singh S, Singla R (2017) Classification of defective modules using object-oriented metrics. *Inter J Intell Syst Technol Appl*. <https://doi.org/10.1504/IJISTA.2017.081311>
- Song Q, Jia Z, Shepperd M, Ying S, Liu J (2011) A general software defect-proneness prediction framework. *IEEE Trans Softw Eng* 37(3):356–370
- Fukushima Takafumi, Kamei Yasutaka, McIntosh Shane, Yamashita Kazuhiro, Ubayashi Naoyasu (2014) An empirical study of just-in-time defect prediction using cross-project models. *Proc MSR Proc Work Conf Min Softw Repos 2014*:172–181
- Satapathy SC, Avadhani PS, Udgata SK, Lakshminarayana S (2013) ICT and critical infrastructure. In: *Proceedings of the 48th annual convention of CSI*, vol 2