Software Defect Prediction based on Classification Rule Mining

Bhushan Bharti[1], Narendra Parmar[2] and Deepak Pathak[3]

Department of Computer Science and Engineering
Sri Satya Sai College of Engineering, RKDF University, Bhopal, India
[1]bhushanbharti76@gmail.com
[2] narendrapcst@gmail.com
[3] deep_325@yahoo.com

* Corresponding Author: Bhushan Bharti Manuscript Received: Manuscript Accepted:

Abstract: **S**oftware development has seen a tremendous growth in recent years. And just like any other thing every software has several defects. These defects may be of different types and different nature depending upon their locations. These defects or bugs can be eliminated greatly during the testing phase. The testing phase is one of the most important phased of the development life cycle of software. If the defects are timely detected and acted upon it can save a lot of time, cost and resources of the development team. Here in this paper we have used Classification Rule Mining as a tool for prediction of software defects and the results are compared with previous research in this field.

Keywords: *Defect Prediction, Classification, Rule Mining*

I. Introduction

Any Software is said to be good software if it is defect free. Software with defects serves no purpose in the industry and in our lives. The demand for quality software is increasing with time. So it becomes very challenging and important task for the software developers to develop defect and error free software. Defects in software systems continue to be a major Challenge [1].Software defect prediction is an important aspect in this regard. The most important aspect of software defect prediction and its elimination is the knowledge of the location of the defect. Software defect prediction is the process of locating defective modules in software. Machine learning classification algorithm is an accepted technique for software fault prediction [2]. A defect can be corrected only if the developer knows its exact location. And it is very important to detect the location and rectify the errors or defects during the development phase of the software because once implemented it does take extra manpower, resources and thus increases the cost of the software. Most commonly the defects lie in different smaller components of the software, so it is a crucial task for the testing team to identify the location and eliminate the errors. If the locations of the possible defects are identified during the development phase, it becomes very easy to get rid of them.

II. Related Work

Our main aim shall be to discuss the work done in the area of software defect prediction and finding out possible solutions to avoid or correct those defects and faults. This is important because software with defects is largely considered as poor quality software.

In 2006, Bibi, Tsoumakas, Stamelos, Vlahavas, applied an approach of machine learning to the defects estimation problem which they called as the Regression via Classification (RvC) [3].The whole process of Regression via Classification (RvC) comprises two important stages

1. It is a method in which the problem of classification is turned into a problem of classification. The target values are converted into classes using a process of discretization.
2. The class output is then reversed to find out the prediction in numerical form.

Menzies, Greenwald, and Frank (MGF) [4] suggesting in their research in 2007 where Rule Induction and Naïve Bayes machine learning algorithms were compared and their performances were analyzed to pre determine the possible defects in components of software under study.

In 2007, MLT(Multilayer Perceptron), Voting feature Intervals(VFI) and NB were used by Oral and Bener [5] for prediction of Embedded Software Defects using seven sets of data for their research In 2007, Iker Gondra [6] used a machine learning methods for defect prediction. He used Artificial neural network as a machine learner.

In 2011, CBA i.e. Classification based Association was used or prediction of Defects in software by Baojun, Karel [7]. The rules generated for CBA-RG classification Association Rules

In [8], predictive models are estimated based on various code attributes to assess the likelihood of software modules containing errors. Many classification methods have been suggested to accomplish this task.

Ahmet Okutan [9] have used Bayesian networks to determine the probabilistic influential relationships among software metrics and defect proneness. The software metrics used in Promise data repository, define two more metrics, i.e. number of developers for the number of developers and lack of coding quality for the source code quality. Mrinal

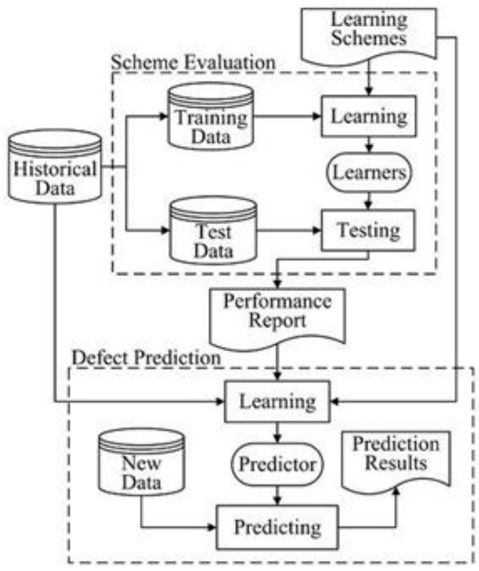
Singh Rawat et al [10] identified causative factors which in turn suggest the remedies to improve software quality and productivity.

III. Proposed Methodology

To construct a model for prediction of software defects, we must first finalize which algorithm or scheme of learning can be used to build that model. This is to ensure the predictive capability of the prediction model that we propose to construct. These steps are very crucial, avoiding which may result in poor performing prediction model with non-reliable prediction and decisions. As a consequence, we use a software defect prediction framework that provides guidance to address these potential shortcomings.

The framework consists of two components:

1. Scheme Evaluation
2. Defect Prediction



3.1 Scheme Evaluation

The scheme evaluation is a fundamental part of the software defect prediction framework. At this stage, different learning schemes are evaluated by building and evaluating learners with them. The first problem of scheme evaluation is how to divide historical data into training and test data. As mentioned above, the test data should be independent of the learner construction. This is a necessary precondition to evaluate the performance of a learner for new data. Cross-validations usually used to estimate how accurately a predictive model will perform in practice. One round of cross-validation involves partitioning a data set into complementary subsets, performing the analysis on one subset, and validating the analysis on the other subset. To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds.

3.1.1 Scheme Evaluation Algoritm

```
Data: Historical Data Set
Result: The mean performance values
    1 M=12 :No of Data Set
    2 i=1;
```

```
3 while i<=M do
4 Read Historical Data Set D[i];
5 Split Data set Instances using % split;
6 Train[i] =60% of D; % Training Data;
7 Learning (Train[i], scheme);
8 Test Data=D[i]-Train[i]; % Test Data;
9 Result=TestClassier(Test[i],Learner);
10 end
```

3.2 Defect Prediction

The defect prediction required extensive research and study of the system and the system components under the probation. The process of the defect prediction is not so easy as it seems. It consists of two stages. The first is the construction of prediction model with a predictor and the second stage is the defect prediction. The defect prediction part of our framework is straightforward; it consists of predictor construction and defect prediction. The prediction model construction involves.

1. A training scheme is build using the historical data and previous performances
2. A predictor is then constructed which helps in the training of data for prediction of defects using previous learning and historical data. This step involves various small tasks like the preprocessing of data and several other tasks for making the data set ready for evaluation of any prediction. The data and the code is then validated and evaluated at several steps for any defect or bug. The predictor plays a very vital role in this process.

After the predictor is constructed, new data are preprocessed in same way as historical data, then the constructed predictor can be used to predict software defect with preprocessed new data.

3.3 Data Set

We used the data taken from the public NASA MDP repository, which was also used by MGF and many others, e.g., [11], [12], [13], [14].Thus, there are 12 data sets in total from NASA MDP repository. Table 4.1 and 4.2 provides some basic summary information. Each data set is comprised of a number of software modules (cases), each containing the corresponding number of defects and various software static codes attributes. After preprocessing, modules that contain one or more defects were labeled as defective. A more detailed description of code attributes or the origin of the MDP data sets can be obtained from.

IV. Result Discussion

We have compared the accuracy and preciseness of our framework with that of others on different parameters and attributes. Comparison in respect of different classification algorithms have also been described below.

4.1 Accuracy

Below is a table 4.1, which is for accuracy. Here we can see different classification algorithms have provided different level of accuracy on different data set. But the overall performances of all are almost same.

Methods	NB	LOG	DT	JRip	OneR	PART	J48	J48G
CM1	83.94	87.68	**89.13**	86.23	**89.13**	73.91	86.23	86.96
JM1	81.28	**82.02**	81.57	81.42	79.67	81.13	79.8	79.83
KC1	83.05	**86.87**	84.84	84.84	83.29	83.89	85.56	85.56
KC3	77.5	71.25	75	76.25	71.25	81.25	80	**82.5**
MC1	**94.34**	99.27	99.25	99.22	99.3	99.19	99.3	99.3
MC2	66	66.67	56.86	56.86	56.86	**70.59**	52.94	54.9
MW1	79.25	77.36	85.85	86.79	85.85	**88.68**	85.85	85.85
PC1	88.82	92.11	**92.43**	89.14	91.45	89.8	87.83	88.49
PC2	94.29	99.05	**99.37**	99.21	**99.37**	**99.37**	98.9	98.9
PC3	34.38	**84.67**	80.22	82.89	82.89	82.67	82.22	83.56
PC4	87.14	**91.79**	90.18	90.36	90.18	88.21	88.21	88.93
PC5	96.56	96.93	97.01	**97.28**	96.9	96.93	97.13	97.16

Table 4.1 Accuracy

4.2 Sensitivity

The performance of NB Tree algorithm is apparently better from the other algorithms as visible in the accuracy table 4.2 below.

The other algorithms have almost an average overall performance.

Methods	NB	LOG	DT	JRip	OneR	PART	J48	J48G
CM1	**0.4**	0.267	0	0.2	0.133	0.333	0.2	0.2
JM1	**0.198**	0.102	0.07	0.157	0.109	0.03	0.131	0.123
KC1	**0.434**	0.238	0.197	0.328	0.254	0.32	0.32	0.32
KC3	**0.412**	**0.412**	0.118	0.118	0.176	0.353	0.353	0.353
MC1	**0.548**	0.161	0.194	0.161	0.161	0.194	0.161	0.161
MC2	**0.571**	0.545	0	0	0.091	0.5	0.045	0.045
MW1	**0.429**	0.286	**0.429**	0.143	.071	0.286	0.214	0.214
PC1	0.28	0.24	0.16	0.16	0.08	**0.36**	0.24	0.24
PC2	**0.333**	0	0	0	0	0	0	0
PC3	**0.986**	0.178	0	0.233	0.014	0.137	0.288	0.288
PC4	0.431	0.538	0.231	0.508	0.323	0.677	**0.692**	0.677
PC5	0.427	0.308	0.332	**0.521**	0.303	0.474	0.498	0.479

Table 4.2 Sensitivity

4.3 Specificity

Below is the table for specificity where some algorithms are showing almost 100% which is only ideal condition and cannot be treated for reality. These predictions of such algorithms may be misleading and wrong.

Methods	NB	LOG	DT	JRip	OneR	PART	J48	J48G
CM1	0.893	0.951	1	0.943	**0.984**	0.789	0.943	0.951
JM1	0.956	0.988	0.99	0.968	0.957	**0.994**	0.954	0.956
KC1	0.898	**0.976**	0.959	0.937	0.932	0.927	0.947	0.947
KC3	0.873	0.794	0.921	0.937	0.857	0.937	0.921	**0.952**
MC1	0.947	**1**	0.999	0.999	**1**	0.999	**1**	**1**
MC2	0.724	0.759	**1**	**1**	0.931	0.862	0.897	0.931
MW1	0.848	0.848	0.924	**0.978**	**0.978**	**0.978**	0.957	0.957
PC1	0.943	0.982	**0.993**	0.957	0.989	0.946	0.935	0.943
PC2	0.946	0.997	**1**	0.998	**1**	**1**	0.995	0.995
PC3	0.219	0.976	0.958	0.944	**0.987**	0.96	0.926	0.942
PC4	0.929	0.968	**0.99**	0.956	0.978	0.909	0.907	0.917
PC5	0.983	0.99	**0.991**	0.987	0.99	0.985	0.986	0.987

Table 4.3 Specificity

4.4 Comparison with other's result

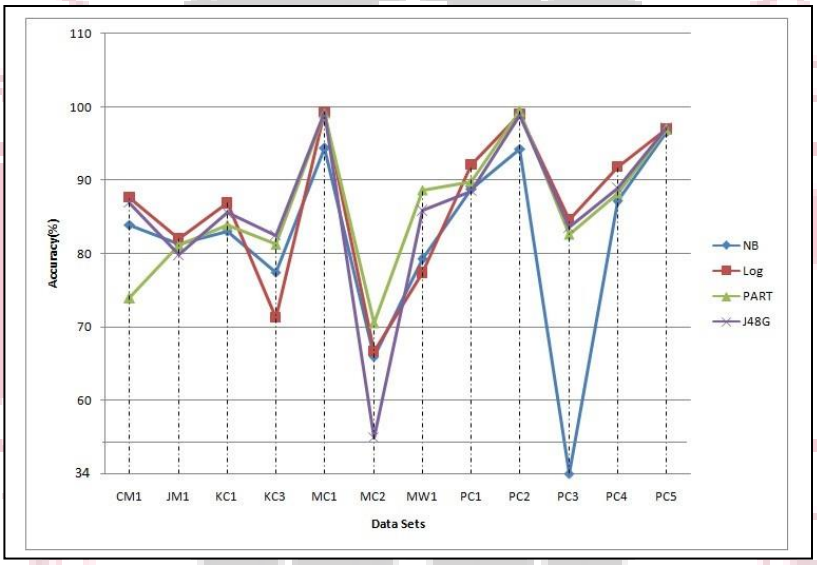


Figure 4.1 Accuracy

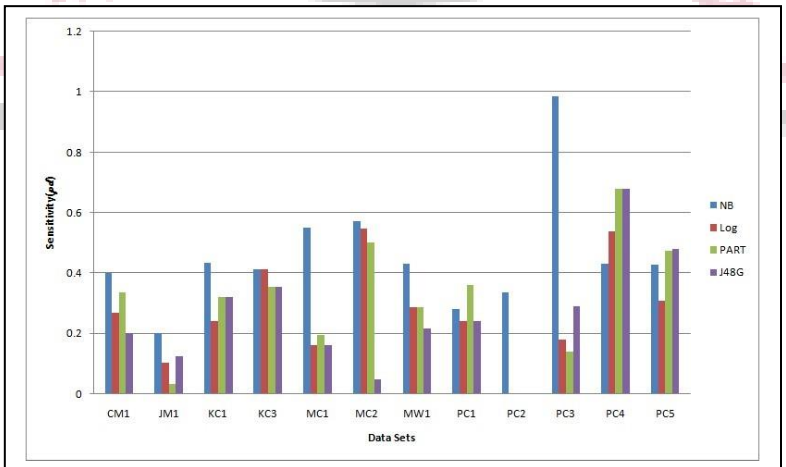


Figure 4.2 Specificity

III. Conclusion & Future Work

In our research work we have attempted to solve the Software defect prediction problem through different Data mining (Classification) algorithms. In our research NB and Logistic algorithm gives the overall better performance for defect prediction. PART and J48 gives better performance than OneR and JRip . From these results, we see that a data preprocessor/attribute selector can play different roles with different learning algorithms for different data sets and that no learning scheme dominates, i.e., always outperforms the others for all data sets. This means we should choose different learning schemes for different data sets, and consequently, the evaluation and decision process is important. In order to improve the efficiency and quality of software development, we can make use of the advantage of data mining to analysis and predict large number of defect data collected in the software development.

References

[1] Parvinder S. Sandhu, Sunil Khullar, Satpreet Singh, Simranjit K. Bains, Manpreet Kaur, Gurvinder Singh, "A Study on Early Prediction of Fault Proneness in Software Modules using Genetic Algorithm", World Academy of Science, Engineering and Technology, 2010, pp. 648-653.

[2] Jang, J-S. R., (1993), “ANFIS-Adaptive-Network Based Fuzzy Inference System”, IEEE Transactions on Systems, Man and Cybernatics, 23(3), pp 665-685.

[3] S Bibi, G Tsoumakas, I Stamelos, and I Vlahavas. Software defect prediction using regression via classification. In IEEE International Conference on, pages 330–336, 2006.

[4] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. Software Engineering, IEEE Transactions on, 33(1):2–13, 2007.

[5] Ata¸c Deniz Oral and Ay¸se Ba¸sar Bener. Defect prediction for embedded software. In Computer and information sciences, 2007. iscis 2007. 22nd international symposium on, pages 1–6. IEEE, 2007.

[6] Iker Gondra. Applying machine learning to software fault-proneness prediction. Journal of Systems and Software, 81(2):186–195, 2008.

[7] Ma Baojun, Karel Dejaeger, Jan Vanthienen, and Bart Baesens. Software defect prediction based on association rule classification. Available at SSRN 1785381, 2011.

[8] M.C.M. Prasad, L.Florence,A.Arya,” A Study on Software Metrics based Software Defect Prediction using Data Mining and Machine Learning Techniques International Journal of Database Theory and Application Vol.8, No.3 (2015).

[9] Okutan, Ahmet, and Olcay Taner Yıldız. "Software defect prediction using Bayesian networks." Empirical Software Engineering 19.1 (2014) 154-181

[10] Mrinal Singh Rawat, et. al.,(2012), “Software Defect Prediction Models for Quality Improvement: A Literature Study”, IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 5, No 2,

[11] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. Software Engineering, IEEE Transactions on, 34(4):485–496, 2008.

[12] Yue Jiang, Bojan Cukic, and Tim Menzies. Fault prediction using early lifecycle data. In Software Reliability, 2007. ISSRE’07. The 18th IEEE International Symposium on, pages 237–246. IEEE, 2007.

[13] Yue Jiang, Bojan Cuki, Tim Menzies, and Nick Bartlow. Comparing design and code metrics for software quality prediction. In Proceedings of the 4th international workshop on Predictor models in software engineering, pages 11–18. ACM, 2008.

[14] Hongyu Zhang, Xiuzhen Zhang, and Ming Gu. Predicting defective software components from code complexity measures. In Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium on, pages 93–96. IEEE, 2007.