Section A-Research paper
ISSN 2063-5346

An Investigation on Classification Accuracy in Software Defect Prediction

Medhunhashini DR*, Dr KS Jeen Marseline
Department of ICT and Cognitive Systems
Sri Krishna Arts and Science College
Coimbatore, India
medhun.hashini@gmail.com

Abstract

An active study field in software engineering is software defect prediction. Before the testing phase even begins, the defect-prone modules are identified using the defect prediction approach. An effective defect prediction model utilizes a few software metrics in order to improve defect prediction. Metrics-based modules enhance software quality, cut costs, and enable efficient resource allocation. Employing classification of defect data with a classifier will increase the effectiveness of defect prediction. Software metrics like Halstead metrics, McCabe's metrics, and LOC based metrics of each module is measured and recorded as a dataset. In this study a real time software project dataset KC1 is taken from NASA Metrics Data Program. Naive Bayes algorithm, Support Vector Machine algorithm, K Nearest Neighbour algorithm and NB Simple algorithm are used as classifiers. The classifications performance is measured using Exactness, Accuracy, Recollection and F Measure. This paper concludes for the used defect dataset an accuracy of 98.89 is obtained with Support Vector Machine algorithm as the best classifier.

Keywords: Software Defect Prediction, Software Metrics, Classifiers, Accuracy

1.Introduction

Software engineering is an active industry where everyday humans need is demandingly converted into software with high reliability. Robust software reduces time and effort spent by the customers. There can be several reasons for the failure of a software functionality. One among them is software defect. Early defect detection in the software development life cycle improves the software's quality.. All facets of software production are addressed by the field of software engineering. Software engineers should take a methodical and organised approach to their work and employ the right tools and approaches depending on the issue at hand, the limitations of the development process, and the available resources. Software engineering is concerned with every step of the software development process, from the initial phases of system specification to the system's maintenance after it has been put to use. The quality of software is described as compliance with openly stated functional and performance objectives, explicitly specified development standards, and implicit qualities that are anticipated of any professionally built software. Profound categorization of defect modules is needed. Simple definition of a defect is "Flaws in the software development process that would result in the software failing to satisfy the desired expectations." [1].
phases on which each phase has entry and exit criteria. The entire framework is suitable for the process management. SDLC has the below defined framework phases.

2.Literature Review

Majority of the reliable and robust software happens identifying the defect at the early stage. This is possible only when a proper classification method is used in the initial stage. There are researchers who proved the efficiency of the software defect prediction incorporating various classification models. The impact of the defect, the risk, and the dependency connected to the projected or forecasted flaws and the extension of the work to assess the various defect prediction methods described by Shihab [2].

Gayatri et al [3] described a sharp increase in the need for software quality estimation. As a result, testing-related difficulties are becoming quite important. Reliability, Functionality, Fault Proneness, Reusability, and Comprehensibility are the SQA qualities for software. Defect prediction or fault proneness is a crucial concern among these. It can be used to gauge the standard and level of client satisfaction, as well as the quality of the final product.

The SDP Methods are those that make use of test data to forecast future software flaws. Software metrics and the software's defect-prone modules are correlated with one another. Software metrics are evaluated in this work as independent variables throughout the Software Development Life Cycle (SDLC), while faulty or non-faulty software is evaluated as a dependent variable was proposed by Prasad et al [4]

Jing et al. [5] described the classification and prediction of software defects namely the Cost-sensitive Discriminative Dictionary Learning (CDDL) approach. The performance of all the comparative methods was assessed using the widely used datasets from the NASA projects as test data. The experimental results demonstrated that, in comparison to a number of representative state-of-the-art defect prediction methods, the CDDL methodology was superior.

Wang et al. [6] used a classification method generated from the data of prior development projects, classifiers divide modules that are characterized by a set of software complexity

metrics, code attributes or features into defect prone and non-defect prone groups. He used the size of the code, complexity of Halstead, and cyclomatic complexity of McCabe's. because they are well-known indicators of software difficulty.

3. Methodology

This section explains the few classification methods Naïve Bayes Classifier, Support Vector Machine, K Nearest Neighbour and NB Simple. Software Product Metrics are used in these classifiers are also discussed. The classifiers Naïve Bayes, K Nearest Neighbour, Support Vector Machine and NB Simple are used on the dataset KC1.The performance of the defect prediction classifiers are evaluated. KC1 a public dataset specifically consist software metric-based methods for the prediction of software defect found in the NASA Metrics Data Program (MDP).

Each collection of data corresponds to a software system or subsystem used by NASA. The dataset includes fault statistics and static code metrics for each associated module. This dataset is used to evaluate a method's prospective real-world performance. KC1 dataset implements with a Storage Management System for collecting and processing ground station data. It includes McCabe and Halstead features including code extractors and module-based controls.[7]

3.1 Software Metrics

Program complexity and software development time estimates have been made using software metrics. A lot of research done to try and find an answer to the hot question, "How to anticipate the quality of software through software metrics, before it is being deployed?" Numerous publications that support statistical techniques and measures that claim to address the quality issue are available. Software metrics typically clarify quantitative measurements of the software product or its requirements Rawat et al. [7] During Developmental Life Cycle (SDLC), software metrics are monitored as independent variables, with faulty or non-faulty software being the dependent variable. The Software Defect Prediction Methods are those that make use of test data to forecast future software flaws. Software metrics and the software's defect-prone modules are interrelated with one another.[8]. Software metrics are of two types Product Metrics and Process Metrics. Product metrics are used for measurement in different stages of Developmental Life Cycle of the software (SDLC). The below table describes the software metrics used in this investigation in specific to KC1 dataset of ground data.

3.1.1 Lines of Code

The standard metric for calculating programme size is lines of code. As a result, it measures the software's size. It is used as a metric to determine the degree of programme complexity and can be calculated in a number of ways, including total lines, lines with comments, lines with executable code, and lines that include both code and comments for each module.

Table 1 LOC Metrics used in KC1 dataset

Metric Used	Description
LOC	Lines of Code
LOCode	Total number of comment lines
LOComment	Total no of executable codes
LOBlank	Total number of blank lines
LOCode & Comment	Total number of Lines of code and comment

3.1.2 McCabe Complexity Metrics

3.1.2.1 Cyclomatic Metrics

A Control Flow Graph is developed a metric used to assess the program's sequential independent routes. It aids in determining the program's complexity. The programme statement is represented by nodes, while its flow is represented by edges. Generally given as V(g) where E is the edge and N is the node.

$$V(G) = E - N + 2$$

3.1.2.2 Essential Metrics

It is the metric where the D-Structured prime sub flowgraph is eliminated, thus reducing the flowgraph. One entrance and one exit sub flowgraph make up the D-Structured prime flowgraph, which is a graph. By removing the primary flowgraph, this measure aids in the identification of unstructured flowgraphs. This complexity is evaluated using. m determines the sub-flow graphs

$$ev(G) = V(G) - m$$

3.1.2.3 Design Metrics

Design Complexity eliminates decisions and nodes that do not affect the calling control over a module's immediate subordinates. It determines the number of decision logic in subroutine calls. Consequently, measuring the interaction between the subroutines is helpful. Denoted as iv(g).

Table 2 McCabe's Metrics used in KC1 dataset

McCabe Complexity Metrics	
v(g)	cyclomatic complexity
ev(g)	essential complexity
iv(g)	design complexity

3.1.3 Halstead Complexity Metrics

Based on the operands and operators employed in the programme, Halstead metrics measures the programme.

Table 3 Halstead's Metrics used in KC1 dataset

Halstead Metrics	
n	Total no of operators + operands
v	Halstead volume

l	Halstead Program level
d	Halstead difficulty
i	Halstead intelligence
e	Halstead effort
b	Halstead error estimate
t	Halstead's time estimator
uniq_Op	unique operators
uniq_Opnd	unique operands
total_Op	total operators
total_Opnd	total operands
branchCount	branch count

3.2 Classification Methods

3.2.1 Naïve Bayes (NB)

Defect prediction is treated as binary classification in an NB technique. It analyses past software module data to train and build a predictor, and it will decide whether or not the new module has flaws depending on the prediction. A software module is selected to serve as the training and prediction object unit by the Naive Bayes Prediction (NBP) approach. "A software module is a programme unit that is discrete and identifiable with regard to compilation and combining with other units," according to the IEEE definition. The Module is also a logically separable part of a program.[9] Naïve Bayes classification algorithm follows Bayes Rule:

$$P(X_1 \ldots . Xn|Y) = \prod_{i=0}^{n} P(X1|Y)$$

3.2.2 K-Nearest Neighbour (KNN)

The voting system is the foundation for how this classifier operates. With the use of previously identified data samples, referred to as the nearest neighbour, and samples that are assigned using the voting procedure, KNN locates new or unidentified data samples. The classification of the data sample involves participation from more than one nearest neighbour. KNN is known as a "lazy learner" since it has a slow rate of learning. KNN is a clustering and classification algorithm. A newly reported problem is classified using a 1-Nearest Neighbour classifier based on the severity of the most similar report from the training set.[10] KNN classifier follows the Euclidean distance given as

$$d(X,Y) = \sqrt{\sum_{i=1}^{n}(Y_i - X_i)^2}$$

The k Nearest Neighbour classification algorithm identifies a group of k entities in the data used for training the dataset that are near to the input and classifies it using the majority of that group's class. [11]

3.2.3 Support Vector Machine (SVM)

A hyper-plane is identified in the input space to divide the sample data into two classes. It maximises the distinction between the classes. By utilising the kernel function theory, SVM performs effectively for the linearly inseparable categorization of data samples. There are many kernel functions that can be used to map data samples to higher dimension feature spaces, such as Gaussian, Polynomial, and Sigmoid. The data samples for the various classes are then divided using a hyper-plane determined by SVM feature space.[12] For the categorization of linearly and nonlinearly separable data, this is a preferable option. SVM boosts its capacity to generalise by adhering to the Structural Risk Minimization (SRM) concept and reducing risk during training. The data points is given as below equation

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3) \ldots \ldots \ldots \ldots, (x_n, y_n)\}$$

By dividing hyperplane, it can see this training data that takes

$$w.x + b = 0$$

3.2.4 NB Simple

A Naive Bayes classifier, where the normal distribution is used to model the numerical properties. A continuous value has a probability distribution depicted as normal distribution. The same has values that are symmetrically distributed largely around the mean [13].

3.2.4.1 Evaluation Methods

Performance metrics can be used to assess how well the defect prediction method is working. Accuracy, Recall, Precision, and F-Measure are the performance measures employed in this work.

Confusion Matrix

A visual of confusion matrix forms the basis for the performance assessment for the classifiers

	PREDICTED	
ACTUAL	True Positive (T_p)	False Negative (F_n)
	False Positive (F_p)	True Negative (T_n)

Accuracy

It is the proportion of accurately predicted faulty modules to all predicted modules. The result is projected as a percentage.

$$Accuracy = \frac{T_p + T_n}{T_p + T_n + F_p + F_n} * 100$$

Recall

It is the percentage of all defect prone modules that were accurately forecasted as units. It goes by the name of sensitivity.

$$Recall = \frac{T_p}{T_p + F_n}$$

Precision

It is the percentage of all forecasted defective modules that were accurately identified as defective units.

$$Precision = \frac{T_p}{T_p + F_p}$$

F-Measure

It is an evaluation of the precision of a model on a dataset.

$$F - Measure = \frac{2 * (Precision * Recall)}{Precision * Recall}$$

Table 5 Confusion Matrix of Classifiers for Defect Prediction

	NB Simple Classifier		Naïve Bayes Classifier		Support Vector Machine		K Nearest Neighbour	
	PREDICTED VALUES							
ACTUAL	440	84	505	19	521	3	518	6
	18	91	70	39	4	105	9	100

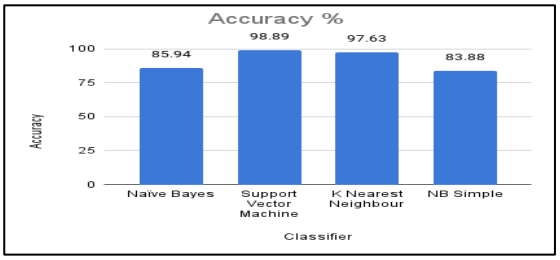
4. Results and Discussion

The experimental results show the classifiers Naïve Bayes has generated an accuracy percentage around 86 whereas the Support Vector Machine has given 98.8 percentage. Comparatively the KNN has proved with accuracy level 97.6 percentage and NB Simple has given 83.8 lowest performance among other three on the data given. Accuracy, Recall, Precision and F-Measure is calculated using Weka Tool. The below table shows the outcome of the classification prediction accuracy.

Table 5 Performance Measures of Classifiers on KC1 Dataset

Classifier	Correctly Classified Instances	Accuracy	Precision	Recall	F Measure
Naïve Bayes	544	85.94	0.843	0.859	0.841
Support Vector Measure	626	98.89	0.989	0.989	0.989
K Nearest Neighbour	618	97.63	0.976	0.976	0.976
NB Simple	531	83.88	0.885	0.839	0.852

Figure 1 Performance Accuracy of Classifiers



5.Conclusion

This paper is developed to understand the best classifier for the data classification on the defect data. Classifiers like Naïve Bayes, Support Vector Machine, K Nearest Neighbour and NB Simple were tested on the KC1 dataset. The preliminary study with the performance measures like accuracy, recall, precision, and F-Measure were calculated and analysed in order to assess the defect prediction's performance. The performance accuracy measure has proved Support Vector Machine does a best classification with an accuracy of 98.89 %. Further this study can be extended with the best classifier SVM the necessary feature can be selected by optimizing to find the best fit using fitness function.

6.References

[1] Kumaresh, S & Baskaran, R 2010, 'Defect analysis and prevention for software process quality improvement. International Journal of Computer Applications, vol. 8, no. 7, pp. 42-47.

[2] Shihab, E 2014, 'Practical software quality prediction', IEEE International Conference on Software Maintenance and Evolution, pp. 639-644.

[3] Gayatri, N, Nickolas, S, Reddy, AV, Reddy, S & Nickolas, AV 2010, Feature selection using decision tree induction in class level metrics dataset for software defect predictions', In Proceedings of the World Congress on Engineering and Computer Science vol. 1, pp. 124-129.

[4] Prasad, MC, Florence, L & Arya, A 2015, '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, pp. 179-190.

[5] Jing, XY, Ying, S, Zhang, ZW, Wu, SS & Liu, J 2014, 'Dictionary learning based software defect prediction' 36th ACM International Conference on Software Engineering, pp. 414-423.

[6] Shirabad, JS & Menzies, TJ 2005, 'The Promise repository of software engineering databases', School of Information Technology and Engineering, University of Ottawa, Canada, vol. 24

[7] Rawat, MS & Dubey, SK 2012, 'Software defect prediction Models for quality improvement: a literature study', International Journal of Computer Science, vol. 9, no. 2, pp. 288-296

[8] Prasad, MC, Florence, L & Arya, A 2015, '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, pp. 179-190

[9] Wang, T & Li, WH 2010, 'Naive bayes software defect prediction Model', International Conference on Computational Intelligence and Software Engineering, pp. 1-4.

[10] Lamkanfi, A, Demeyer, S, Soetens, QD & Verdonck, T 2011, 'Comparing mining algorithms for predicting the severity of a reported bug', 15th European Conference on Software Maintenance and Reengineering, pp. 249-258.

[11] Sridhar & Babu, 2012, 'Evaluating the Classification Accuracy of Data Mining Algorithms for Anonymized Data', IEEE Transactions on Software Engineering, vol. 3, no. 8, pp. 63- 67

[12] Xing, F, Guo, P & Lyu, MR 2005, 'A novel method for early software quality prediction based on support vector machine', 16th IEEE International Symposium on Software Reliability Engineering, pp. 10-17

[13] Wang, J, Shen, B & Chen, Y 2012a, 'Compressed C4. 5 Models for Software Defect Prediction', 12th International Conference on Quality Software, pp. 13-16