Recent trends in Management and Commerce
Vol: 1(3), 2020
REST Publisher; ISSN: 978-81-936097-6-7
Website: http://restpublisher.com/book-series/rmc/

Software Engineering Defect Prediction using the SPSS Method

Moolpani Deepak Inder
SSt College of Arts and Commerce, Maharashtra, India.
*Corresponding Author Email: deepak.moolpani@sstcollege.edu.in

Abstract

Software Deficiency (SDP) is among the most helpful the SDLC test step. It distinguishes volumesDeficiency is required and requires detailed testing.In this way, test sources can beUsed efficiently withoutExceeds the restrictions.Software Errors has emerged as a prominent research avenue inside this field of software engineering. Developers can detect potential issues and defects using the program's flawed predictions improvingTests the sourcesThe program improves reliability.The best way to observe and test each activity of the application and run the testSoftware, check the real softwareDevices to help you understand what problems may be for the final userGo and check the softwareDifferent test environment, briefly, notice and resolve its simple test and if notmodels for software faults are employed to automatically identify problematic classes prior to system testing. These models can lower the cost of resources, infrastructure, and the test period. To enhance the test procedure, we suggest a new defect prediction model in this study. Software systems are now more expansive and intricate than ever. It can be quite difficult to prevent software diseases with such characteristics. In order to effectively allocate scarce resources, it is required to automatically estimate the quantity of errors in software. Several methods have been put forth to quickly and cheaply find and fix these flaws. The effectiveness of these strategies needs to be significantly improved, though.In order to estimate the quantity of software system flaws, we therefore suggest a novel approach in this work that makes use of deep learning techniques. First, companies are processing the public database, which includes data normalisation and registration changes. In the interests of preparing the required data for a machine learning model, we perform the data sampling second. Lastly, we feed summary statistics to a deep neural network that has been specifically created to anticipate the number of flaws. We test the suggested strategy in two reputable datasets as well. The findings of the assessment show that the suggested approach is precise and can be improved in more advanced approaches. Ratio studies are statistical analyses of data from appraisals and property valuations. Nearly all states utilise them to produce quantitative measure of the proportion of current market price about which individually estimated taxable property is appraised as well as to offer assessment performance indicators. Software defect prediction, Deep learning, Software quality, Software metrics and Robustness evaluation The Cronbach's Alpha Reliability result. The overall Cronbach's Alpha value for the model is .658 which indicates 66% reliability. From the literature review, the above 50% Cronbach's Alpha value model can be considered for analysis. Software Engineering Defect Predictionthe Cronbach's Alpha Reliability result. The overall Cronbach's Alpha value for the model is .658 which indicates 66% reliability. From the literature review, the above 50% Cronbach's Alpha value model can be considered for analysis.

Keywords: Software defect prediction, Deep learning, Software quality, Software metrics and Robustness evaluation

Introduction

Software quality assurance relies heavily on software defect prediction, one of the most active study fields in the field of software engineering. The expanding complexity and interdependence of programming has raised both the challenge of giving quality, low-cost, and maintained software, along with the risk of creating software flaws. Incorrect or unexpected outputs and behaviours are typical effects of software flaws. The technique of defect prediction is crucial and vital. By identifying flawed components (events) before testing, defect predictors can lower costs and boost software quality, allowing software developers to efficiently optimise the distribution of scarce resources for maintenance and evaluation [1].systems for developing software. Predicting software flaws and their locations, for instance, which modules are more flawed, is a focus. Researchers have investigated a number of methods that can produce a prediction system using known

training instances in order to do this. Experimental validation is required because we lack a thorough theory and it is unclear which strategies are the "best" [2].Predictions of software defects can significantly boost test efficiency and software quality. Defect prediction has already used a number of data mining techniques. We identified many Naive Bayes-based defect predictors and examined their difference estimation techniques and algorithmic complexity. By comparing this model to Decision Tree Learner J48 and doing a predictive performance evaluation, we discovered that non - linear and non-Gauss Naive Bayes (MVGNP) was the most effective. MVGNP has been demonstrated to be beneficial for deficiencies predictions by experimental results on benchmark data sets from MDP [3].It is very difficult to foresee software defects because there are more specimens of defective modules than of working ones. Since many information mining algorithms attempt to increase overall accuracy yet struggle in classes with insufficient sample sizes, they end up producing subpar models in this situation. For instance, an algorithm that consistently forecasts a block as genetic condition will attain the greatest accuracy if the total number of flawed samples surpasses 95% of the problematic samples[4].studies that anticipate software defects Although there are several research articles on the subject, according to Fenton and Neal (1999), the defect prediction problem has not yet been fully resolved. When disabilities are identified or observed, several incorrect assumptions are made, which results in incorrect conclusions. When we consider that some describe flaws as regarding that matter and others as residuals, their claim can be better understood. We can observe from the publications on problem prediction that early studies made considerable use of standard code features. However, additional measurements, such as process measurements, are also helpful and should be looked into aside from the impact of standard document measurement techniques on defect prediction [5].Since then, an astounding number of innovative and even updated methods have been found for software metrics. This was particularly true as investigation into mining application code, sometimes known as "empirical software engineering 2.0," increased. For a number of reasons, we are not trying to criticise empirical programming skills as a whole. Defect prediction, however, is a perfect illustration of empirical software development research where the science community have lost all sense of the forest amidst the numerous trees to be felled [6].In order to comprehend the characteristics of defective blocks, predictive defect models are applied. In the context of a humongous software system, investigate the relationship among both developer-centered organizational change measures and the likelihood of customer-reported defects. Also look at the influence of software on software reliability and indeed the characteristics of increased and surprise defects in workplace circumstances. The association between modern code review procedures, defect-proneness, and software quality Planning programmers for quality improvement requires this understanding [7].Using predictive classification techniques from code properties, software defect prediction seeks to quickly detect fault-prone modules in order to enhance software quality and test efficiency. For this purpose, a number of categorization models have been assessed. However, further study is required to improve convergence throughout studies and further increase confidence throughout test results because there are conflicting results regarding actual performance of one classification and the utility of measurement system classification in general. We take into account three possible sources of bias: making comparisons classifiers on one or a limited number of internally developed sets of data, relying on conceptually unnecessary accuracy indicators for operating system error prediction and bridge comparative, and lastly, sparingly employing statistical testing techniques. reliable empirical results [8].Data for software modules is typically scarce. In this study, we suggest using dictionary learning as a tool for predicting software defects. Using metric characteristics taken from open-source software, we learn several dictionaries (containing defective blocking and non-defective blocking sub-dictionaries and the overall dictionary) and scanty representation coefficients. Additionally, we account for the misclassification cost issue because misclassifying defective batches typically entails a larger risk cost than correctly classifying non-defective ones. We suggest a cost-sensitive racially discriminatory dictionary learning (CDDL) method for identifying and forecasting software defects [9].prediction of software defects. Although asymmetrical learning can enhance prediction performance, overall findings appear to be quite uneven and conflicting. This uncertainty surrounding the application of discrepancy learning for the prediction of software defects, in our opinion, stems from three key factors. First off, standard performance metrics are flawed. Second, the degree of discrepancy in the data on software defects and its impact on forecasting accuracy have not been studied. Third, there is a lack of knowledge regarding the connection between asymmetric learning techniques and classifier selection [10].Software defect assertion: A regression in which a regression model forecasts the quantity of defects and a classifier forecasts the module's class label (eg binary classification: defect or no defect). The ensemble learning procedure for a classification problem entails two steps: (1) training a set of independent classifiers known as base classifiers, and (2) integrating the output of the classification models using aggregating or voting to arrive at a final prediction. Heterogeneous refers to a single type of algorithm when the underlying classifier is made up of various types of algorithms [11].This study looks at software features that aid practitioners in comprehending software faults that have an impact on code quality in addition to anticipating software vulnerabilities for developers and businesses. We also wish to investigate the capability of these traits to forecast software flaws. Our study investigates the research issues that follow in light of this objective. prediction of software defects. First off, standard performance metrics are flawed. Second, the degree of discrepancy in the data on software defects and its relationship to forecasting accuracy have not been investigated. Lastly, the connection between asymmetric learning strategies and classifier selection is not fully understood. The selection of data source and inputs metric kinds is similar [12].prediction of software defects. The SMOTE methodology, which overestimates representations of a minority class by creating representations of the minority class, is the subject of our experiments. A

new instance of the minority class is created by SMOTE at a random location along the feature space line that connects it to its closest neighbours of the same class. By using univariate sampling, we correspondingly undersample the classification results [13].The software history of a project is important since a project's properties could be unstable at first. So because bugs in the most recent modifications have yet to be found and repaired, they are not included. Keep in mind that this is simply an estimate. In actuality, we anticipate recent modifications so that programmers can find mistakes early, as we tested in our research project [14].prediction of software defects. Extracting software systems from free software repositories is the initial step. A software module could be a class, a file, a method, a code modification, etc. Marking the software components as buggy or clean is the next step. Information about bugs is gleaned through post-release flaws reported in bug tracking programmes like Bugzilla. A software module is labelled as deprecated if it has flaws discovered in later releases. Extraction of the utilizing advanced from the application components is the third stage. Character-based, new currency, Formation of advanced glycation end, AST-tree-based, ASTpath-based, and Alanine aminotransferase code characteristics are frequent in deep learning-based software fault prediction [15].software applications. For other systems not employed in studies, our method might yield better or worse results. By choosing software with a variety of functionalities (software platforms, servers, and desktop applications), built in several programming languages, we reduce this hazard [16]."DAASE: Dynamic Adaptation Automated Software Engineering" and "SEBASE: Software Development by Automated Search." The first author's trip to Xidian University in China, funded by an EU FP7 IRSES grant on "NICaiA: Environment Inspired Computing and its Applications," allowed him to finish some of the work (Grant 247619). J.-C. Lu, associate editor. The Institute of Excellence and Development in Cognitive Computing and Applications is where the authors are located [17].to anticipate software flaws. By employing the Wavelet coefficients score sampling approach on the labelled defect-free blocks, we first create a training dataset with labels that are comparable to classes. Then, we compute the relation graph's negative sparse weights, which act as grouping indicators, using the negative sparse technique. Lastly, we iteratively anticipate the labels of unmarked programming blocks in a negatively sparse graph using a label propagation approach. For the purpose of classifying and predicting software defects, we recommend an adverse sparse regression label propagation method that makes extensive use of both labelled and unlabeled data to increase generalizability [18].software undertaking. After the model has been created, the next stage is to gather the information to validate that model; in order to accomplish this successfully, a thorough description of each variable was needed. In Section 3.1, we provide a first-level thorough explanation of the set of criteria. An excerpt from a later survey given to project management team to gather information on finished projects is provided in Section 3.2. Some of the problems with this approach to measuring construction quality factors are discussed in Section 3.3 [19].Techniques for predicting software problems make it possible to find defective software components. The sequence in which the programming should be inspected can be decided using code assessment or unit testing. Developers can then devote their limited development resources to the parts of the code that are the most likely to have problems. The personnel and time costs that are saved as a result can lower total maintenance costs and boost business profitability [20].

Materials and Methods

Software defect prediction: One of the most useful metrics in the test phase of sdlc is software deficiency (SDP). The volume that needs faulty and thorough testing is identified. This allows for the efficient use of test resources while yet respecting barriers. In order to improve software reliability, the study of software defects has grown to be a prominent research area. In hopes of improving the dependability of the programmer, developers employ programmer deficiency predictions to aid in the identification of prospective issues and the improvement of test resources.

Deep learning: Deep learning is a sub -group of mechanical learning, which is basically a neurological network with three or more layers. These neurological networks seek to simulate the behavior of the human brain-although it is not compatible with its ability-it allows you to "learn" from high data. Deep learning is a technological learning method that teaches computers how to learn by doing, just as humans do. A key component of driverless automobiles is deep learning, which makes it easier to recognise stops or tell a pedestrian from a lamppost.

Software quality: A discipline of study and practise known as "software quality" is described as describing the desired characteristics of software products. Software quality can be approached from two different angles: defect prevention and quality attributes. Software that does not contain acceptable errors or defects, is provided promptly and within the allotted budget, and satisfies and upholds requirements and/or expectations is considered to be of high quality. Software quality in the context of software engineering represents both structural and functional quality.

Software metrics: The software metric is the measurement of measured or calculated software properties. Software measurements are valuable for a number of reasons, including software performance, planning work, productivity and many applications. Software measurements include measurement quality, including some quantities of measurements. This can be classified into three categories: product measurements, process measurements and project measurements. In the first part of this blog posting series on measurements, we have reviewed four types of promidius measurements: counters, measurements, histograms and wrinkles.

Robustness evaluation: The strong assessment volume in the ration is simultaneously evaluating a large number of potential bug scenes and providing versatile tools for a detailed analysis of project strength. The current version supports the system and density uncertainty, which is connected to create a group shot. One of the most widely used definitions of

the pattern in the pharmacy is presented by ICH: 'The strongest of an analysis process, which refers to its ability to not be affected by the variations in the parameters, and the method of deliberately.

Method: SPSS Statistics is a statistical control Advanced Analytics, Multivariate Analytics, Business enterprise Intelligence and IBM a statistic created by a software program is a package crook research. A set of generated statistics is Crook Research is for a long time SPSS Inc. Produced by, it was acquired by IBM in 2009. Current versions (after 2015) icon Named: IBM SPSS Statistics. The name of the software program is to start with social Became the Statistical Package for Science (SPSS) Reflects the real marketplace, then information SPSS is converted into product and service solutions Widely used for statistical evaluation within the social sciences is an application used. pasted into a syntax statement. Programs are interactive Directed or unsupervised production Through the workflow facility. SPSS Statistics is an internal log Organization, types of information, information processing and on applicable documents imposes regulations, these jointly programming make it easier. SPSS datasets are two-dimensional Have a tabular structure, in which Queues usually form Events (with individuals or families) and Columns (age, gender or family income with) to form measurements. of records Only categories are described: Miscellaneous and Text content (or "string"). All statistics Processing is also sequential through the statement (dataset) going on Files are one-to-one and one-to-one Many can be matched, although many are not in addition to those case-variables form and By processing, there may be a separate matrix session, There you have matrix and linear algebra on matrices using functions Information may be processed.

Result and Discussion

TABLE 1. Descriptive Statistics

	N	**Range**	**Minimum**	**Maximum**	**Sum**	**Mean**		**Std. Deviation**	**Variance**
Software defect prediction	90	4	1	5	282	3.13	.115	1.093	1.196
Deep learning	90	4	1	5	270	3.00	.131	1.245	1.551
Software quality	90	4	1	5	291	3.23	.133	1.264	1.597
Software metrics	90	4	1	5	294	3.27	.119	1.130	1.276
Robustness evaluation	90	4	1	5	297	3.30	.158	1.495	2.235
Valid N (listwise)	90								

Table 1 shows the descriptive statistics values for analysis N, range, minimum, maximum, mean, standard deviation Software defect prediction, Deep learning, Software quality, Software metrics and Robustness evaluationthis also using.

TABLE 2. Frequencies Statistics

		Software defect prediction	**Deep learning**	**Software quality**	**Software metrics**	**Robustness evaluation**
N	Valid	90	90	90	90	90
	Missing	0	0	0	0	0
Mean		3.13	3.00	3.23	3.27	3.30
Std. Error of Mean		.115	.131	.133	.119	.158
Median		3.00	3.00	3.00	3.00	3.00
Mode		3	3	3	3	5
Std. Deviation		1.093	1.245	1.264	1.130	1.495
Variance		1.196	1.551	1.597	1.276	2.235
Skewness		-.429	.321	-.043	-.260	-.098
Std. Error of Skewness		.254	.254	.254	.254	.254
Kurtosis		.047	-.794	-.900	-.198	-1.484
Std. Error of Kurtosis		.503	.503	.503	.503	.503
Range		4	4	4	4	4
Minimum		1	1	1	1	1
Maximum		5	5	5	5	5
Sum		282	270	291	294	297
Percentiles	25	3.00	2.00	2.00	3.00	2.00
	50	3.00	3.00	3.00	3.00	3.00
	75	4.00	4.00	4.00	4.00	5.00

Table 2 Show the Frequency Statistics in Software Engineering Defect Prediction.Software defect prediction, Deep learning, Software quality, Software metrics and Robustness evaluationcurve values are given.

TABLE 3. Reliability Statistics

Cronbach's Alpha Based on Standardized Items	N of Items
.658	5

Table 3 shows the Cronbach's Alpha Reliability result. The overall Cronbach's Alpha value for the model is .658 which indicates 66% reliability. From the literature review, the above 50% Cronbach's Alpha value model can be considered for analysis.

TABLE 4. Reliability Statistic individual

	Cronbach's Alpha if Item Deleted
Software defect prediction	.587
Deep learning	.656
Software quality	.536
Software metrics	.591
Robustness evaluation	.614

Table 4 Shows the Reliability Statistic individual parameter Cronbach's Alpha Reliability results. The Cronbach's Alpha value for Software defect prediction.587, Deep learning .656, Software quality.536, Software metrics.591 and Robustness evaluation.614 this indicates all the parameter can be considered for analysis.

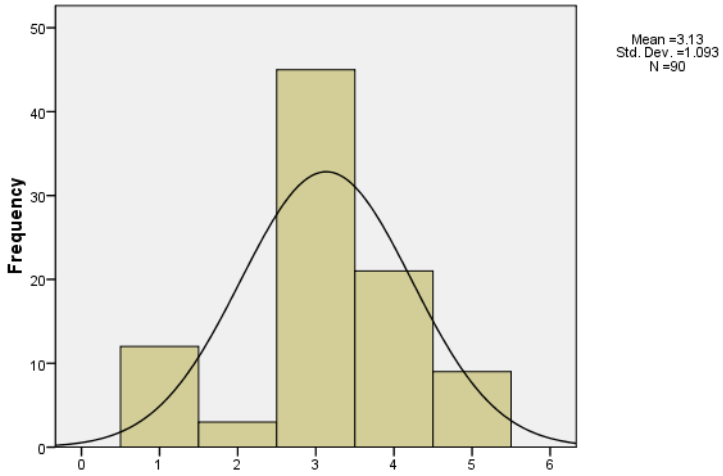

FIGURE 1. Software defect prediction

Figure 1 shows the histogram plot for Software defect prediction from the figure it is clearly seen that the data are slightly Left skewed due to more respondent chosen 3 for Software defect prediction except the 2 value all other values are under the normal curve shows model is significantly following normal distribution.

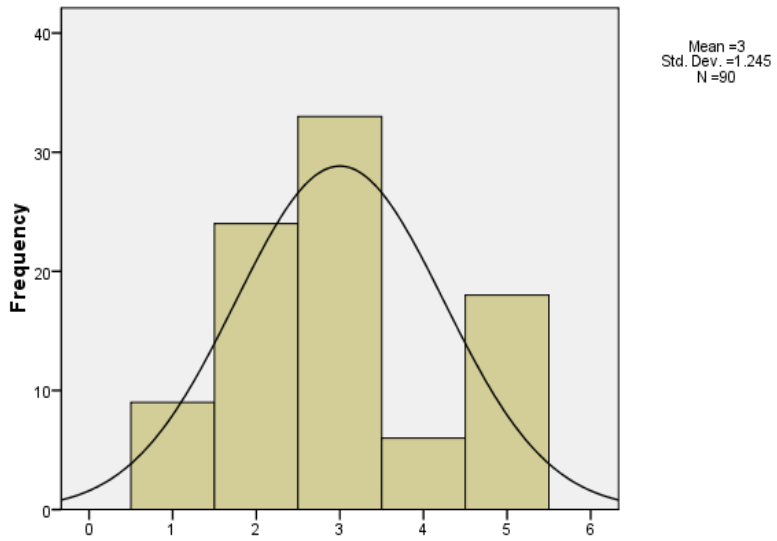

FIGURE 2. Deep learning

Figure 2 shows the histogram plot for Deep learning from the figure it is clearly seen that the data are slightly Left skewed due to more respondent chosen 3 for Deep learning except the 2 value all other values are under the normal curve shows model is significantly following normal distribution.

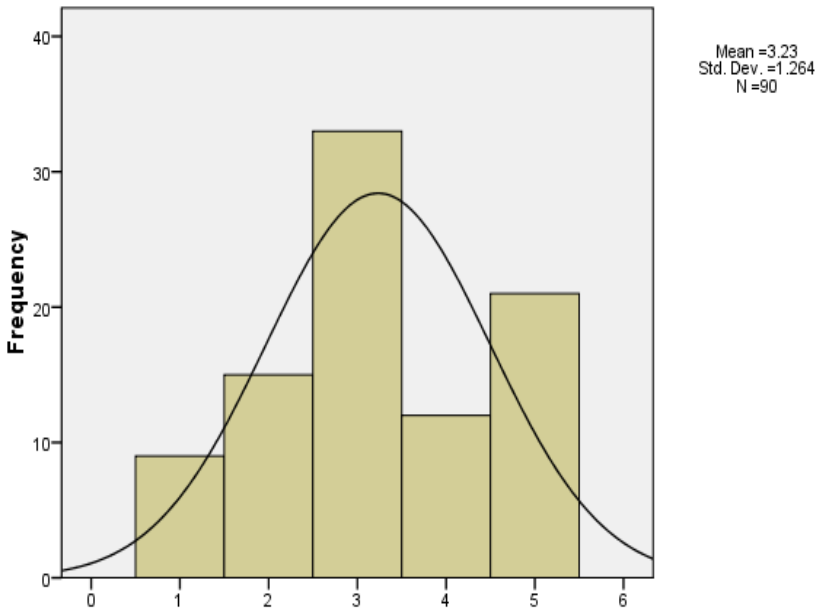

FIGURE 3. Software quality

Figure 3 shows the histogram plot for Software quality from the figure it is clearly seen that the data are slightly Left skewed due to more respondent chosen 3 for Software quality except the 3 value all other values are under the normal curve shows model is significantly following normal distribution.

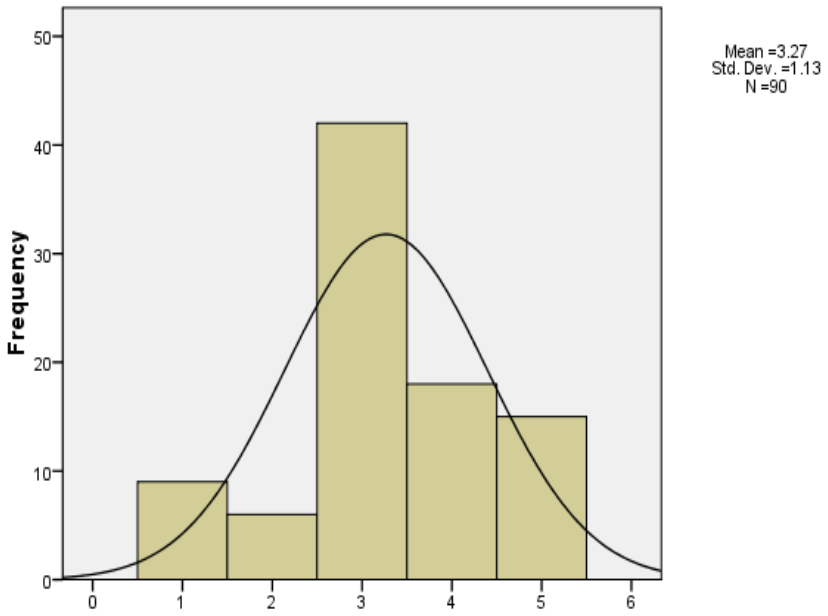

FIGURE 4. Software metrics

Figure 4 shows the histogram plot for Software metricsfrom the figure it is clearly seen that the data are slightly Left skewed due to more respondent chosen 3 for Software metrics except the 2 value all other values are under the normal curve shows model is significantly following normal distribution.

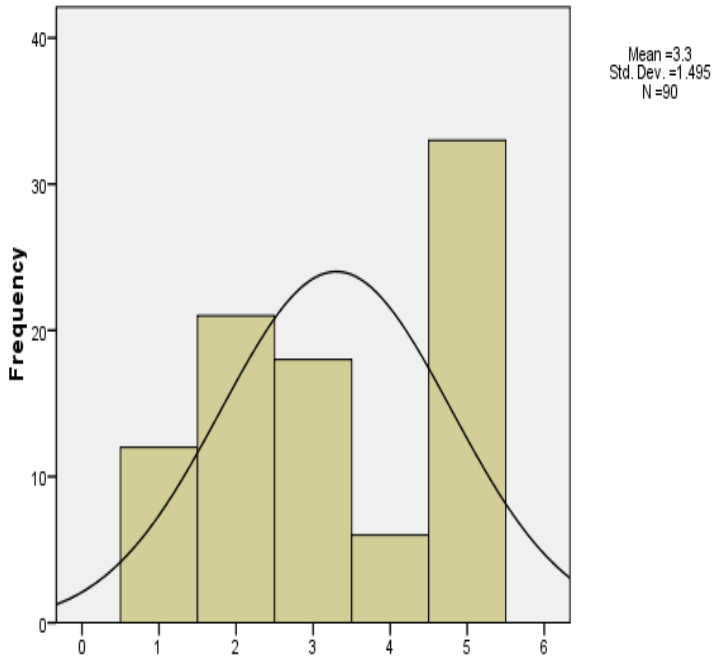

FIGURE 5. Robustness evaluation

Figure 5 shows the histogram plot for Robustness evaluationfrom the figure it is clearly seen that the data are slightly Right skewed due to more respondent chosen 5 for Robustness evaluationexcept the 2 value all other values are under the normal curve shows model is significantly following normal distribution.

TABLE 5. Correlations

	Software defect prediction	**Deep learning**	**Software quality**	**Software metrics**	**Robustness evaluation**
Software defect prediction	1	.149	.368**	.407**	.264*
Deep learning	.149	1	.214*	.096	.290**
Software quality	.368**	.214*	1	.499**	.319**
Software metrics	.407**	.096	.499**	1	.172
Robustness evaluation	.264*	.290**	.319**	.172	1
**. Correlation is significant at the 0.01 level (2-tailed).					
*. Correlation is significant at the 0.05 level (2-tailed).					

Table 5 shows the correlation between motivation parameters for Software defect prediction. For Software metricsis having highest correlation with Deep learning and having lowest correlation.Next the correlation between motivation parameters for Deep learning. For Robustness evaluation is having highest correlation with Software defect prediction and having lowest correlation**.**Next the correlation between motivation parameters for Software quality. For Software metrics is having highest correlation with Deep learning and having lowest correlation**.**Next the correlation between motivation parameters for Software metrics. For Software quality is having highest correlation with Deep learning and having lowest correlation**.** Next the correlation between motivation parameters for Robustness evaluation. For Software qualityis having highest correlation with Software metrics and having lowest correlation**.**

Conclusion

Software Deficiency (SDP) is among the most helpful the SDLC test step. It distinguishes volumes Deficiency is required and requires detailed testing. In this way, test sources can be Used efficiently without Exceeds the restrictions. Software Errors has emerged as a prominent research avenue inside this field of software engineering. Developers can detect potential issues and defects using the program's flawed predictions improving Tests the sources The program improves reliability. The best way to observe and test each activity of the application and run the test Software, check the real software Devices to help you understand what problems Software systems are now more expansive and intricate than ever. It can be quite difficult to prevent software diseases with such characteristics. In order to effectively allocate scarce resources, it is required to automatically estimate the quantity of errors in software. Several methods have been put forth to quickly and cheaply find and fix these flaws. systems for developing software. Predicting software flaws and their locations, for instance, which modules are more flawed, is a focus. Researchers have investigated a number of methods that can produce a prediction system using known training instances in order to do this. Experimental validation is required because we lack a thorough theory and it is unclear which strategies are the "best" Predictions of software defects can significantly boost test efficiency and software quality. Defect prediction has already used a number of data mining techniques. We identified many Naive Bayes-based defect predictors and examined their difference estimation techniques and algorithmic complexity The software metric is the measurement of measured or calculated software properties. Software measurements are valuable for a number of reasons, including software performance, planning work, productivity and many applications. Software measurements include measurement quality, including some quantities of measurements. The strong assessment volume in the ration is simultaneously evaluating a large number of potential bug scenes and providing versatile tools for a detailed analysis of project strength. The current version supports the system and density uncertainty, which is connected to create a group shot. One of the most widely used definitions of the pattern in the pharmacy is presented by ICH: 'The strongest of an analysis process, which refers to its ability to not be affected by the variations in the parameters, and the method of deliberately. Ratio studies are statistical analyses of data from appraisals and property valuations. Nearly all states utilise them to produce quantitative measure of the proportion of current market price about which individually estimated taxable property is appraised as well as to offer assessment performance indicators. Software defect prediction, Deep learning, Software quality, Software metrics and Robustness Evaluation.The Cronbach's Alpha Reliability result. The overall Cronbach's Alpha value for the model is .658 which indicates 66% reliability. From the literature review, the above 50% Cronbach's Alpha value model can be considered for analysis.

REFERENCES

1. Li, Zhiqiang, Xiao-Yuan Jing, and Xiaoke Zhu. "Progress on approaches to software defect prediction." Iet Software 12, no. 3 (2018): 161-175.

2. Shepperd, Martin, David Bowes, and Tracy Hall. "Researcher bias: The use of machine learning in software defect prediction." IEEE Transactions on Software Engineering 40, no. 6 (2014): 603-616.
3. Wang, Tao, and Wei-hua Li. "Naive bayes software defect prediction model." In 2010 International conference on computational intelligence and software engineering, pp. 1-4. Ieee, 2010.
4. Rodriguez, Daniel, Israel Herraiz, Rachel Harrison, Javier Dolado, and José C. Riquelme. "Preliminary comparison of techniques for dealing with imbalance in software defect prediction." In Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, pp. 1-10. 2014.
5. Okutan, Ahmet, and OlcayTanerYıldız. "Software defect prediction using Bayesian networks." Empirical Software Engineering 19 (2014): 154-181.
6. Lanza, Michele, Andrea Mocci, and Luca Ponzanelli. "The tragedy of defect prediction, prince of empirical software engineering research." IEEE Software 33, no. 6 (2016): 102-105.
7. Tantithamthavorn, Chakkrit, Shane McIntosh, Ahmed E. Hassan, Akinori Ihara, and Kenichi Matsumoto. "The impact of mislabelling on the performance and interpretation of defect prediction models." In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 1, pp. 812-823. IEEE, 2015.
8. Lessmann, Stefan, Bart Baesens, Christophe Mues, and SwantjePietsch. "Benchmarking classification models for software defect prediction: A proposed framework and novel findings." IEEE transactions on software engineering 34, no. 4 (2008): 485-496.
9. Jing, Xiao-Yuan, Shi Ying, Zhi-Wu Zhang, Shan-Shan Wu, and Jin Liu. "Dictionary learning based software defect prediction." In Proceedings of the 36th international conference on software engineering, pp. 414-423. 2014.
10. Song, Qinbao, YuchenGuo, and Martin Shepperd. "A comprehensive investigation of the role of imbalanced learning for software defect prediction." IEEE Transactions on Software Engineering 45, no. 12 (2018): 1253-1269.
11. Aljamaan, Hamoud, and AmalAlazba. "Software defect prediction using tree-based ensembles." In Proceedings of the 16th ACM international conference on predictive models and data analytics in software engineering, pp. 1-10. 2020.
12. Esteves, Geanderson, Eduardo Figueiredo, Adriano Veloso, Markos Viggiato, and NivioZiviani. "Understanding machine learning software defect predictions." Automated Software Engineering 27, no. 3-4 (2020): 369-392.
13. Pelayo, Lourdes, and Scott Dick. "Applying novel resampling strategies to software defect prediction." In NAFIPS 2007-2007 Annual meeting of the North American fuzzy information processing society, pp. 69-72. IEEE, 2007.
14. Tan, Ming, Lin Tan, Sashank Dara, and Caleb Mayeux. "Online defect prediction for imbalanced data." In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 2, pp. 99-108. IEEE, 2015.
15. Pan, Cong, Minyan Lu, and Biao Xu. "An empirical study on software defect prediction using codebert model." Applied Sciences 11, no. 11 (2021): 4793.
16. Wang, Song, Taiyue Liu, Jaechang Nam, and Lin Tan. "Deep semantic feature learning for software defect prediction." IEEE Transactions on Software Engineering 46, no. 12 (2018): 1267-1293.
17. Wang, Shuo, and Xin Yao. "Using class imbalance learning for software defect prediction." IEEE Transactions on Reliability 62, no. 2 (2013): 434-443.
18. Zhang, Zhi-Wu, Xiao-Yuan Jing, and Tie-Jian Wang. "Label propagation based semi-supervised learning for software defect prediction." Automated Software Engineering 24 (2017): 47-69.
19. Fenton, Norman, Martin Neil, William Marsh, Peter Hearty, Lukasz Radlinski, and Paul Krause. "Project data incorporating qualitative factors for improved software defect prediction." In Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007), pp. 2-2. IEEE, 2007.
20. Qiao, Lei, Xuesong Li, QasimUmer, and Ping Guo. "Deep learning based software defect prediction." Neurocomputing 385 (2020): 100-110.

