International Journal of Advancement in Electronics and Computer Engineering (IJAECE)
Volume 13, Issue 7, July. 2024, pp.101-105, ISSN 2278 -1412
Copyright © 2012: IJAECE (www.ijaece.com)

INTERPRETABLE ACCURACY OF SOFTWARE DEFECT PREDICTION USING MACHINE LEARNING TECHNIQUE

[1] Monika Nagar, [2] Dr. Indu Shrivastava, [3] Swati Khanve, [4] Dr. Sneha Soni,
[1] M. Tech. Scholar, CSE SIRTE Bhopal, , India
[2,3] Prof., CSE SIRTE Bhopal,swatikhanve55.sk@gmail.com, India
[4] Prof. & Head of Dept., CSE SIRTE Bhopal, soni.snehaa@gmail.com, India

Abstract: - *The presence of software defects can lead to substantial impacts in terms of the functionality, reliability, and overall effectiveness of software systems. The identification and elimination of defects during the initial phases of software develo pment are of the highest priority in ensuring the availability of software products of superior quality. The software defect prediction is to predict the defects in historical data base. So, in real world, it is difficult to predict because it requires more number of data variables, metrics and historical data. The ML concentrates on the algorithms entirely centered on statistical methods and data mining techniques for classifying and predicting the defects and these statistical methods followed are quite similar to regression based methods which we used earlier to the ML. The RF ML technique is providing good accuracy compared to other LR and SVM technique. In this model is simulated python language and calculated simulation parameter i.e. precision, recall and accuracy.*

Keywords: - Software Defects, Accuracy, Precision, Recall, Random Forest (RF), Logistic Regression (LR), Support Vector Machine (SVM)

I. INTRODUCTION

A software defect (SD) refers to a visible discrepancy that may prevent a software system from performing its intended function. This event is usually called a defect identified by a tester and is commonly called a bug/flaw/error [1, 2]. According to the definition of ISO/IEC/IEEE 24765, a software defect can be described as a clear indication of an error in a software system. Software Defect Prediction (SDP) is an effective method that can actually identify software modules or classes that are more likely to be defective [3]. Developers are forced to go through a process of continuous changes, timelines, and the need to ensure flawless functionality in their work [9]. Currently, limited staff and time requirements pose a great challenge in testing software systems. Therefore, it is important to allocate available resources efficiently in favor of source code that may contain more defective modules, and therefore, the allocation of required testing resources. As an example, let us consider the cost of a Java- based project [4] with a size of 100 function points and 10 KLOC. The effort is distributed across five different phases as follows:

10% of the budget is allocated for requirements development, analysis, and design; 20% is used for coding; 30% is devoted to testing; and 5% is allocated for development and training.

To make an accurate prediction, it is necessary to focus on modules that are error-prone and require more extensive testing. This approach helps target and deploy limited testing resources, thereby reducing the overall testing effort. The widespread use of software systems across industries highlights the potentially severe consequences of software failures that can pose serious risks to human health and cause significant economic burdens. For example, up to 200 deaths per year in the UK can be attributed to preventable defects in silent care systems [5].

In the US, the annual financial costs of software failures in software systems are estimated to be between $22.2 billion and $59.5 billion [6]. It is therefore increasingly important to ensure that software systems are of the highest quality. SDP methods typically use a supervised approach, where a set of independent variables (also called predictors) is used to make predictions about dependent variables (in this case error-prone modules). Models are created using a combination of AI techniques [13], deep learning techniques [6], and measurable learning techniques [8].

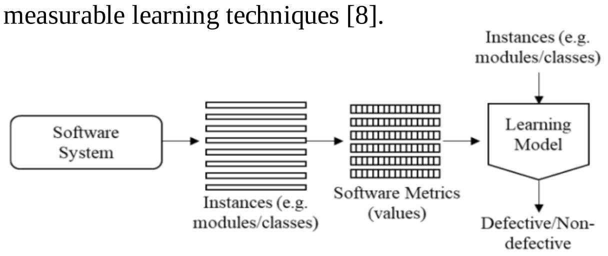

Instances (e.g. modules/classes)
Software System
Instances (e.g. modules/classes)
Software Metrics (values)
Learning Model
Defective/Non-defective

Figure 1: SD Technique

Figure 1 shows the basic design of the SDP method. To avoid programming errors, the first step is to organize a measurement data set using programming modules that contain characteristics such as module size and module complexity. These measurements are usually obtained from a variant control system (VCS). It is also important to assign the proper flag to the broken modules in case they are affected by code changes aimed at fixing a specific issue that falls under the category of deformation.

Deformation models are also created using AI processes. It is also important to have the edge settings of the AI techniques in place properly to control the quality of the AI techniques. A trained model can

DOI Member 135.66.411

predict the classification (faulty or not) of different modules after training is complete. Both potential and brand new projects can benefit from these predictions. There are two different types of training prediction models: homogeneous software defect prediction and heterogeneous software defect prediction. The two types of homogeneous software defect prediction are further categorized. First, there is the within-project defect prediction (WPDP) model, which uses labeled instances from project A to build a prediction model and uses it to classify unlabeled instances in the same project as good or bad. Second, cross-project defect prediction (CPDP) uses labeled instances from project A (called source) to train a prediction model to predict issues in project B (called target).

II. SOFTWARE QUALITY AND SD PREDICTION

II.1 Software Quality

In computer science, software pertains to the processing of data by hardware, programs and other data to obtain information. Software has been described as a program developed in segments with source code. It is a set of agents handling complexity, testability, visibility, changeability, conformity, reliability and traceability [9]. Software has become the foundation of modern technology; it constitutes or controls the products and services that human beings rely on for a wide variety of daily activi ties, from the crucial to the trivial. It therefore refers to measurable characteristics. Juran's defines quality as a product featu re that satisfies customer needs, providing customer satisfaction. The American Society for Quality (ASQ) defines quality as the foremost characteristics of the product and it must endure the quantified and inferred needs and the software distributed to th e user community should be free of defects and scarcities. Software characteristics are measured with respect to its complexity, cohesion, streaks of code, amount of function points and other factors. Quality needs to be measured which imply to deliver a product designed in a stated way. When we design a product, the quality of the software designed must pertains to the feature s specified for that product by the designer [10].

Waman believes that a successful project gives rise to customer satisfaction. It was advocated that meeting customer ex pectations leads to quality. Software quality assurance is achieved through designing test cases and using them as a control measure to ensure desired quality. Quality Assurance of the Software is defined as a list of events intended to appraise the product tha t will also help us to develop and maintain the software [11, 12].

II.2 Software Defect Prediction

Product lifecycle is a complex and critical process that enables the creation of error-free software and ensures that software is free of defects. This crucial step involves identifying and correcting defects even before the final product is delivered to the customer. Software memory contains a wealth of data that is essential to determine the quality of a product. These large data stores are essential to ensure error-free software standards and provide an optimal user experience to customers. By applying AI-based algorithms and data mining techniques to such documents, we aim to extract key information while correcting defects and improving the overall quality of the product. A product defect is an error, mistake, omission, flaw, or oversight in the programming or design of a computer system. This leads to inaccuracies or unexpected things and as a result, unintended behavior.

The impact of software bugs can be significant, affecting both the cost and the quality. Furthermore, identifying and correcting these errors can be very costly. To address this error pattern, monitoring teams should: Focus on minimizing errors and improving software quality. Additionally, some attention should be given to training and developing professional practices aimed at minimizing and correcting such errors. This important approach will bring significant benefits in terms of improved software performance and overall organizational success. Software error detection is a crucial step that helps to identify and correct faulty software modules. Ensuring high quality of software and minimizing minor errors results in a reliable and efficient product. Detecting and correcting errors at an early stage significantly reduces development costs and improves software performance. This strategy not only promotes cost efficiency, but also leads to the creation of more robust and sophisticated software, which benefits both developers and end users in the long run.

SDP can also be used as a factor in planning programs in both the intelligent world and in enterprises. Credits from previous iterations of a product are added to the static code, along with a log of the various transformations used to create models to replace damaged modules in the next delivery of the product. This makes it an area of such parts Programming that is likely to contain consequences is useful. This is useful when trying to deal with expenditures limited to the entire program structure that are too large to deal with completely. There are good indicators of adaptation, decision making to control product planning while focusing on testing products and their defective parts. SDP goes beyond the domains of business and science. One aspect of SDP is: ability to address software planning in both academic and business settings. Developers can create models to predict potential issues in subsequent versions by incorporating static code concepts that collect logs of various changes and extract relevant information from previous software versions. This proactive methodology helps to identify and resolve imperfections in specific modules. It ultimately improves the overall quality of the product and contributes to the quality of the product.

In today's fast-paced digital environment, the importance of efficient cost and resource management in software development cannot be overemphasized. The scope of the entire software structure is often enormous, so careful planning and focus on tasks are essential to avoid burnout and succeed. Furthermore, the presence of potential defects further emphasizes the need for careful evaluation to identify flaws and defects in the product design and correct them appropriately. Focusing on a comprehensive and detailed testing approach, including careful consideration of weak areas, will ensure a solid and error-free end result.

III. PROPOSED METHODOLOGY

A system that is used to understand the concept and its environment using a simplified interpretation of the environment usin g model is called cognitive system. The step that we pass to construct the model is known as inductive learning. The Cognitive system is able to combine its experience by constructing new structures is patterns. The constructed model and pattern by cognitive system is called machine learning. Models that are described as predictive since it can be used to predict the outp ut of a function (target function) for a given value in the function„s domain while informative pattern are characterized only describes the portion of data.

Random Forest is an ensemble method based on principle of bagging. It uses decision trees as base classifiers. To generate each single tree in Random Forest, Breiman followed following steps: If the number of records in the training set is N, then N records are sampled at random but with replacement, from the original data; this is bootstrap sample.

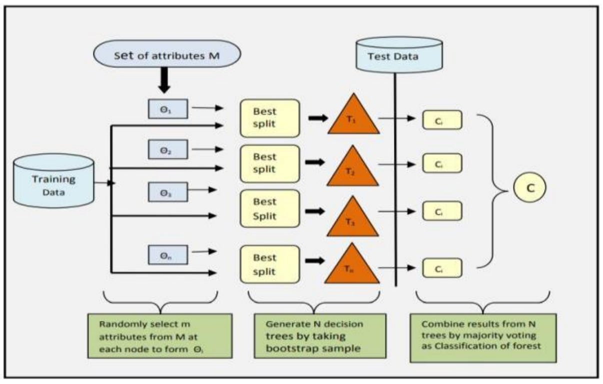

Set of attributes M
Test Data
Training Data
Θ₁
Θ₂
Θ₃
Θₙ
Best split
Best split
Best Split
Best split
T₁
T₂
T₃
Tₙ
Cᵢ
Cᵢ
Cᵢ
Cᵢ
C
Randomly select m attributes from M at each node to form Θᵢ
Generate N decision trees by taking bootstrap sample
Combine results from N trees by majority voting as Classification of forest

Figure 2: Random Forest classifier.

This sample will be the training set for growing the tree. If there are M input variables, a number m << M is selected such t hat at each node, m variables are selected at random out of M and the best split on these m attributes is used to split the node. The value of m is held constant while the forest is growing. Each tree is grown to the largest possible extent. There is no pruning. In this way, multiple trees are induced in the forest; the number of trees is pre-decided by the parameter Ntree. The number of variables (m) selected at each node is referred to as mtry or k in the literature. The depth of the tree can be controlled by a paramet er nodesize (i.e. number of instances in the leaf node), which is usually set to one. Once the forest is trained or built as explained above, to classify a new instance, it is run across all the trees grown in the forest. Each tree gives classification for the new instance which is recorded as a vote. The votes from all trees are combined and the class for which maximum votes are counted (majority voting) is declared as the classification of the new instance.

IV. SIMULATION RESULT

In this test case, we considered other standard classification scheme such as SVM, LR and RF classifier.

Parameter:
Accuracy gives a proportion of how precise your model is in anticipating the real up -sides out of the absolute up-sides anticipated
by your framework. Review gives the quantity of real up-sides caught by our model by grouping these as obvious positive. F- measure can give a harmony among accuracy and review, and it is liked over precision where information is uneven.

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

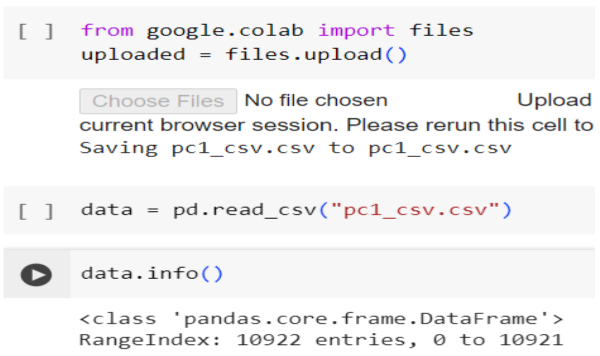
$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Re call} = \frac{TP}{TP + FN}$$

Where,
TP = True Positive,

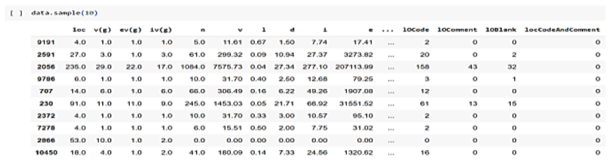
TN = True Negative FP = False Positive, FN = False Negative

Data Information:
CM1 dataset contains total 11141 entries which are given below



```
[ ]  from google.colab import files
     uploaded = files.upload()

     Choose Files  No chosen file                    Upload
     current browser session. Please rerun this cell to
     Saving pc1_csv.csv to pc1_csv.csv

[ ]  data = pd.read_csv("pc1_csv.csv")

     data.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 10922 entries, 0 to 10921
```

Data Sample:



```
[ ] data.sample(10)

         loc  v(g)  ev(g)  iv(g)       n        v     l      d       i          e  ...  lOCode  lOComment  lOBlank  locCodeAndComment
9191     4.0   1.0    1.0    1.0     5.0    11.61  0.67   1.50    7.74      17.41  ...       2          0        0                  0
2591    27.0   3.0    1.0    3.0    61.0   299.32  0.09  10.94   27.37    3273.82  ...      20          0        2                  0
2056   235.0  29.0   22.0   17.0  1084.0  7575.73  0.04  27.34  277.10  207113.99  ...     158         43       32                  0
9786     6.0   1.0    1.0    1.0    10.0    31.70  0.40   2.50   12.68      79.25  ...       3          0        1                  0
707     14.0   6.0    1.0    6.0    66.0   306.49  0.16   6.22   49.26    1907.08  ...      12          0        0                  0
230     91.0  11.0   11.0    9.0   245.0  1453.03  0.05  21.71   66.92   31551.52  ...      61         13       15                  0
2372     4.0   1.0    1.0    1.0    10.0    31.70  0.33   3.00   10.57      95.10  ...       2          0        0                  0
7278     4.0   1.0    1.0    1.0     6.0    15.51  0.50   2.00    7.75      31.02  ...       2          0        0                  0
2866    53.0  10.0    1.0    2.0     0.0     0.00  0.00   0.00    0.00       0.00  ...       0          0        0                  0
10450   18.0   4.0    1.0    2.0    41.0   180.09  0.14   7.33   24.56    1320.62  ...      16          0        0                  0
```

Classifier Technique:

```python
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

Table I: Comparison Result

Classifier	Precision	Recall	Accuracy
SVM	99.34%	96.98%	96.90%
LR	99.56%	99.24%	98.56%
RF	100%	99.89%	99.86%

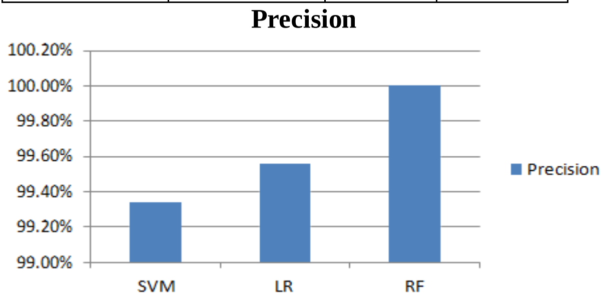


Figure 3: Graphical Represent of Precision

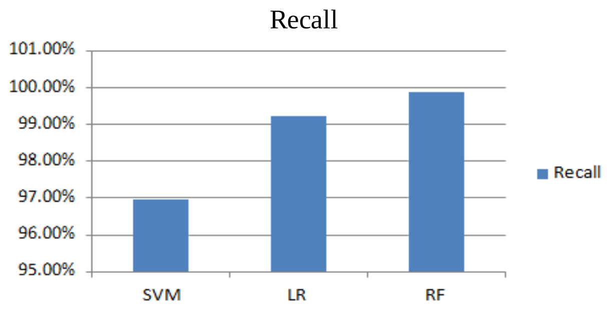


Figure 4: Graphical Represent of Recall

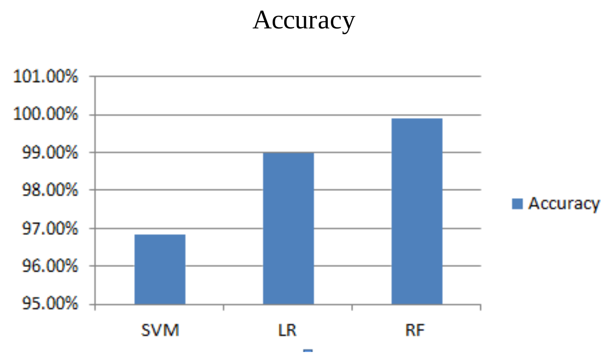


Figure 5: Graphical Represent of Accuracy

Table II displays the results of Muhammad Azam et al. [1] and implemented method in terms of accuracy. Malarvizhi et al. [1]
give an accuracy of 93.05% for SVM, 93.32% for LR and 93.86% for RF. The implemented method provides an accuracy of
96.84% for SVM, 98.99% for LR and 99.90% for RF. Clearly, the implemented method is a 6.04% improvement accuracy compared to Muhammad Azam et al. [1]. Fig. 6 shows the graphical representation of the comparison result.

Table II: Comparison Results

Design	Method	Accuracy
Muhammad Azam et al. [1]	SVM	93.05%
	LR	93.32%
	RF	93.86%
Implemented Design	SVM	96.84%
	LR	98.99%
	RF	99.90%

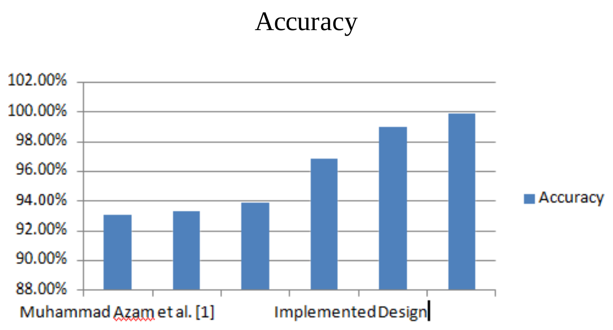


Figure 6: Graphical Represent of Accuracy method

V. CONCLUSION

In this paper By utilizing SDP, organizations can proactively identify and address potential software flaws, resulting in improved software performance and customer satisfaction. The application of this methodology enables companies to prioritize testing efforts an d allocate resources efficiently, ultimately leading to enhanced software reliability and reduced risks. Malarvizhi et al. [1] give an accuracy of 93.05% for SVM, 93.32% for LR and 93.86% for RF. The implemented method provides an accuracy of 96.84% for SVM, 98.99% for LR and 99.90% for RF. Clearly, the implemented method is a 6.04% improvement accuracy compared to Muhammad Azam et al. [1].
.

REFERENCES

[1] Muhammad Azam, Muhammad Nouman and Ahsan Rehman Gill, "Comparative Analysis of Machine Learning techniques to Improve Software Defect Prediction", KIET Journal of Computing &

Information Sciences [KJCIS], vol. 5, no. 2, pp. 41-66, 2022.

[2] L.-Q. Chen, C. Wang, and S.-L. Song, „„Software defect prediction based on nested-stacking and heterogeneous feature selection,"" Complex Intell. Syst., vol. 8, no. 4, pp. 3333–3348, Aug. 2022

[3] M. Pavana, L. Pushpa, and A. Parkavi, „„Software fault prediction using machine learning algorithms,"" in Proc. Int. Conf. Adv. Elect. Comput. Technol., 2022, pp. 185–197

[4] A. Al-Nusirat, F. Hanandeh, M. K. Kharabsheh, M. Al-Ayyoub, and N. Al-Dhfairi, „„Dynamic detection of software defects using supervised learning techniques,"" Int. J. Commun. Netw. Inf. Secur., vol. 11, no. 1, pp. 185–191, Apr. 2022.

[5] R. Bahaweres, F. Agustian, I. Hermadi, A. Suroso, and Y. Arkeman, „„Software defect prediction using neural network basedSMOTE,"" in Proc. 7th Int. Conf. Electr. Eng., Comput. Sci. Informat. (EECSI), Oct. 2020, pp. 71–76

[6] N. Li, M. Shepperd, and Y. Guo, „„A systematic review of unsupervised learning techniques for software defect prediction,"" Inf. Softw. Technol., vol. 122, Jun. 2020, Art. no. 106287.

[7] Y. Qiu, Y. Liu, A. Liu, J. Zhu, and J. Xu, „„Automatic feature exploration and an application in defect prediction,"" IEEE Access, vol. 7, pp. 112097–112112, 2019.

[8] A. Alsaeedi and M. Z. Khan, „„Software defect prediction using supervised machine learning and ensemble techniques: A comparative study,"" J. Softw. Eng. Appl., vol. 12, no. 5, pp. 85–100, 2019.

[9] C. Manjula and L. Florence, „„Deep neural network based hybrid approach for software defect prediction using software metrics,"" Cluster Comput., vol. 22, no. S4, pp. 9847–9863, Jul. 2019.

[10] R. Jayanthi and L. Florence, „„Software defect prediction techniques using metrics based on neural network classifier,"" Cluster Comput., vol. 22, no. S1, pp. 77–88, Jan. 2019.

[11] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, „„Software bug prediction using machine learning approach,"" Int. J. Adv. Comput. Sci. Appl., vol. 9, no. 2, pp. 78–83, 2018.

[12] N. Kalaivani and R. Beena, „„Overview of software defect prediction using machine learning algorithms,"" Int. J. Pure Appl. Math., vol. 118, pp. 3863–3873, Feb. 2018.

[13] M. A. Memon, M.-U.-R. Magsi, M. Memon, and S. Hyder, „„Defects prediction and prevention approaches for quality software development,"" Int. J. Adv. Comput. Sci. Appl., vol. 9, no. 8, pp. 451–457, 2018.

[14] E. Naresh and S. P. Shankar, „„Comparative analysis of the various data mining techniques for defect prediction using the NASA MDP datasets for better quality of the software product,"" Adv. Comput. Sci. Technol., vol. 10, no. 7, pp. 2005–2017, 2017.

[15] D. Kumar and V. H. S. Shukla, „„A defect prediction model for software product based on ANFIS,"" Int. J. Sci. Res. Devices vol. 3, no.

10, pp. 1024–1028, 2016.

[16] P. Mandal and A. S. Ami, „„Selecting best attributes for software defect prediction,"" in Proc. IEEE Int. WIE Conf. Electr. Comput. Eng., Dec. 2015, pp. 110–113.

[17] M. C. M. Prasad, L. F. Florence, and A. Arya, „„A study on software metrics based software defect prediction using data mining and machine learning techniques,"" Int. J. Database Theory Appl., vol. 8, no. 3, pp. 179–190, Jun. 2015.

[18] A. Chug and S. Dhall, „„Software defect prediction using supervised learning algorithm and unsupervised learning algorithm,"" in Proc. 4th Int. Conf. Confluence Next Gener. Inf. Technol. Summit, Noida, 2013, pp. 173–179.