International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7461
***Article Received**: 20 May 2023 **Revised:** 15 July 2023 **Accepted:** 02 August 2023*

An Innovative Approach for Predicting Software Defects by Handling Class Imbalance Problem

Ranjeetsingh Suryawanshi[1], Amol Kadam[2], Devata Anekar[3], Vinayak Patil[4]
[1,2] Department of Computer Engineering, College of Engineering, Bharati Vidyapeeth Deemed To Be University Pune, Maharashtra, India
[3]Sinhgad academy of engineering,Savitribai Phule Pune University,Maharashtra, India
[4]Bharati vidyapeeth college of engineering Navi Mumbai, Maharashtra, India
ranjeetsinghsuryawanshi@gmail.com
akkadam@bvucoep.edu.in
devanekar@gmail.com
vinayak.n.patil@bvcoenm.edu.in

Abstract— From last decade unbalanced data has gained attention as a major challenge for enhancing software quality and reliability. Due to evolution in advanced software development tools and processes, today's developed software product is much larger and complicated in nature. The software business faces a major issue in maintaining software performance and efficiency as well as cost of handling software issues after deployment of software product. The effectiveness of defect prediction model has been hampered by unbalanced data in terms of data analysis, biased result, model accuracy and decision making. Predicting defects before they affect your software product is one way to cut costs required to maintain software quality. In this study we are proposing model using two level approach for class imbalance problem which will enhance accuracy of prediction model. In the first level, model will balance predictive class at data level by applying sampling method. Second level we will use Random Forest machine learning approach which will create strong classifier for software defect. Hence, we can enhance software defect prediction model accuracy by handling class imbalance issue at data and algorithm level.

Keywords- Software defect prediction;Data imbalance;Machine Learning;Prediction model;Sampling.

I. INTRODUCTION

The software business faces a constant struggle in building open source and commercial software projects with zero bugs. Due to the limited number of frameworks that have been developed, there is no standardization of the software defect prediction process. The software fault forecasting framework performs well on training data but averagely on testing data since it was not trained with a varied variety of data. We must focus on the quality of the dataset if we want to obtain high accuracy in the defect prediction model. When enough data is provided to develop the prediction model, a higher degree of defect prediction accuracy is achieved. The scenario, though, may differ from project to project. For instance, the initial release of new software does not contain any flaw information which makes difficult for prediction model as no historical data available.

In high dimensional datasets, it is very difficult to extract important features from large number of data features. More features or dimensions in dataset can reduce accuracy of prediction model because it deals with huge number of dimensions and it is known as Curse of dimensionality. B. Pes conducted research to investigate the efficiency of hybrid learning strategies that integrate dimensionality reduction approache for resolving class imbalance problem [1].

In some software defect dataset has a class imbalance issue, due to this defect prediction models are likely to select the class that contains larger samples which cost decreases accuracy of prediction model. Issue of class imbalance can be resolved by under-sampling and over-sampling technique which helps to improve accuracy of model. Under-sampling techniques removes instances of the majority class and over-sampling techniques increases instances of the minority class present in defect prediction dataset. For developing prediction model there are many machine learning techniques available which are:

Logistic regression:[2] In binary classification logistic regression is very effective method. It produces a highly robust discriminative model that is based on function.

$$f(y) = \frac{1}{(1 + e^{-y})}$$

Support vector machine: It is a distinct classifier that uses a suitable hyperplane as the decision boundary to differentiate data from two classes Typically, the hyperplane is used to maximize the distance between classes. It generates optimum hyperplane by solving a Lagrangian optimization problem [3].

Decision tree: A decision tree machine algorithm is used for both regression and classification problems. The decision tree

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7461
***Article Received**: 20 May 2023 **Revised:** 15 July 2023 **Accepted:** 02 August 2023*

method divides the data recursively into subsets depending on the most significant attribute at each node of the tree. [4].

Random forests: It comprise a set of decision tree, in which every tree is dependent on random vector values collected independently. Mathematically It can be defined as $\{h(x,\theta_k)\}, k = 1,\dots\}$ where input x is represented by a set of random vectors $\{\theta_k\}$, each of which cast unit vote for most preferred class. Breiman applied the randomization technique to generate diversity among base decision trees, which works satisfactorily with bagging or random subspace approaches. [5][6].

Naive Bayes: Naive Bayes is appropriate for huge datasets since it is predicated on the idea that particular variables are independent of one another and that such a model does not need recurrent estimations of parameters. [7].

k-nearest neighbour: It identifies new samples according to the minimum distance in the initial data. KNN determines the correct class by measuring the dispersion between the test data and all of the training points[7].

This study is organized systematically as follows: section I gives an introduction to the research problem. Section II describes corresponding research carried out for handling class imbalance issue. Section III describes the proposed architecture for predicting software defects by addressing the class imbalance problem. In Section IV we discussed what we found for our research questions. We conclude our final remark in conclusion segment.

II.RELATED WORK

Although many researchers and academician made excellent contributions, still H. Krasner's (2018) quality report estimates that the cost of software reliability for exterior defects and breakdowns is roughly around 635 billion dollars. According to Report, fixing software flaws can lower quality costs[8]. According to research articles it suggests that this can be minimized by addressing class imbalance issue, removing outliers in classifications, and handling high dimension data. We have completed literature survey of available research articles for software defect prediction using machine learning approach[9].In this work we proposing novel approach for software fault prediction by handling class imbalance problem.

Research on software defect prediction has started in the international arena since early 1970s.Akiyama found a correlation between the number of flaws and the program's judgement calls after discovering a total of 546 bugs across 9 modules during study observation [10]. A. Ihara et al solved bug issue in upcoming release of bug fix. This work performed experiment on Eclipse software and the findings of the experiment demonstrate that the next release of open-source software can fix bug problem[11].

Qiao Yu et al used an attribute selection and attribute ranking approach to solve the issue of fault prediction performance for dissimilar project. The work tested on the NASA and PROMISE datasets and the findings shows that performance has been enhanced using this method for cross-project defect prediction [12]. N. Nagwani et al work finds expert to fix software bugs by mapping a list of expert developers with common bug keywords. The system will forecast experienced developers for a specific bug by mapping frequent phrases with developer relationships. 1000 software bugs from the Mozilla bug repository and 97 developers are selected in the experiment for matching bug expert[13].

Santosh Singh Rathore et al study concentrated on establishing a link between fault proneness of object-oriented software systems and class level object-oriented metrics with the purpose of evaluating coupling, cohesion, complexity, and inheritance design features[14]. With the use of test case execution paths within the code, Prateek Anand solved the problem of future flaws being released due to change in code. Twelve data sets from four industrial projects were used to verify defect prediction, and the top 10 faulty features were predicted with an average normalised cumulative gain of 0. 684[15]. Shruthi Puranik et al. devised an approach to forecast the fault sensitivity score using minimal R square values and regressions as intermediate stage. This work tested using Eclipse JDT Core dataset [16]. Santosh Singh Rathore et al used five-fold cross-validation method to test the work on the PROMISE data repository for intra-release prediction. Older releases are utilized as the training dataset and testing dataset for inter-releases prediction[17].

Jianming Zheng et al investigated that accuracy of software defect prediction model depends upon balanced distribution of defect data sets. This study demonstrates higher prediction accuracy over the minority class by using two unique algorithms that gain knowledge from unbalanced data sets. In first approach IAdaBoost method is used which classifies imbalanced data sets using adaptive cost matrix. In the Second approach, the SWIMBoost technique creates minority samples by using the oversampling method and it increases decision boundaries of the minority class of unbalanced data which helps to boost classification accuracy[18].

Ruchika Malhotra et al handled issue of imbalance dataset and dimensionality reduction using naïve bays classifier with SMOTE sampling technique. support vector machine synthetic minority oversampling technique focuses on borderline area between minority and majority classes, so misclassification is avoided at border area. Linear Discriminant Analysis is used for

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7461
***Article Received**: 20 May 2023 **Revised:** 15 July 2023 **Accepted:** 02 August 2023*

dimensionality reduction by calculating variance and the distance between means of the two classes[19]. Santosh Singh Rathore et al focused on the issue of segmenting the defect dataset into various module subsets, each of which is trained with a different learning technique before combining the results. The PROMISE and the Eclipse bug data set used to test this concept and the results show that the dynamic preferred strategy gives good result[20].

Thanh Tung Khuat et al performed experiment on imbalanced data with synthetic minority oversampling technique and ensembles classifiers (LR,SVM,Decision Tree,KNN).This work experimentally analyses the significance of data sampling in software defect prediction model using different classifiers on imbalanced data.Experimental result shows that combined sampling approaches using base classifiers gives improved defect prediction performance and generates much better F1-score values[21]. Somya Goyal proposed Neighbourhood based undersampling approach for prediction defect in software with high accuracy. This algorithm avoid information loss in dataset by getting more visibility of minority data points and minimizing the surplus elimination of majority data points while sampling dataset.[22]. Maohua Gan et al proposed research idea to solve the issue of class imbalance which is primarily based on the negative and positive values of data set which is used to compute accuracy measures. This research provides a rating of predictions across multiple data sets using the proposed measures, which can discriminate between successful and failed forecasts[23].

P. Soltanzadeh et al. developed an enhanced SMOTE-based technique, Range-Controlled SMOTE (RCSMOTE), to overcome the overgeneralizing problem caused by oversampling of noisy data and overlapping between various classes near class boundaries. This work is carried out in three phases. 1) splitting up the unbalanced data input into major and minor classes. 2) finding noisy minority class instances, and 3)preventing generation of minority instances inside majority class regions[24].

F. Thabtah et al explained two approaches for solving class imbalance problem 1. Data driven: This approach balances the class distribution with training dataset for e.g. Oversampling and undersampling technique. 2.Algorithm driven: This approach balances the class distribution using machine learning algorithms without training dataset for e.g. cost sensitive learning (cost of misclassification), thresholding.This work performed experiment with Nave Bayes as base classifier and five datasets from the University of California Irvine (UCI) repository[25]

The researchers came up with several methods to handle the class imbalance issue. We are proposing an approach to boost the prediction model accuracy by addressing the class imbalance problem with software fault detection. From our proposed design we will be able to answer following research questions (RQ):

RQ1. How to increase the accuracy for software defect prediction model?

RQ2. How to decrease the impact of class imbalance on software defect prediction?

RQ3. How to handle high dimensional dataset used for software defect prediction

III. PROPOSED FRAMEWORK FOR SOFTWARE DEFECT PREDICTION

Figure 1 shows proposed framework for handling class imbalance issue in software fault prediction. It consists of following stages:

- Data extraction and pre-processing
- Handling high-dimensional data
- Data Sampling
- Building SDP Model
- Performance measures

Data extraction and pre-processing:

For our experimental setup we will be using NASA dataset[26]. We extracted data from CM1 and JM1 dataset for data preprocessing. Firstly, we explored CM1 dataset which contains 39 features and 344 observations with Majority class contain 302 observations whereas Minority class contain 42 observations. We also explored CM1 data for defective class distribution and from figure 2 it shows that there is imbalanced data for defective and non-defective class. First, we need to balance class distribution in order get better result in data balancing stage.

Handling high-dimensional data:

More features or dimensions in dataset can reduce accuracy of prediction model, so we are using gini impurity, entropy and information gain to handle high dimensional data.

For CM1 data there are total 39 features, due to this high dimension data we have calculated gini impurity for Feature selection using following mathematical equation.

$$\text{Gini Impurity} = 1 - \sum_{i=1}^{c}(p_i)^2 \quad \ldots\ldots\ldots\ldots\ldots (1)$$

Table 1 shows gini impurity calculated for CM1 dataset.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7461
***Article Received**: 20 May 2023 **Revised:** 15 July 2023 **Accepted:** 02 August 2023*

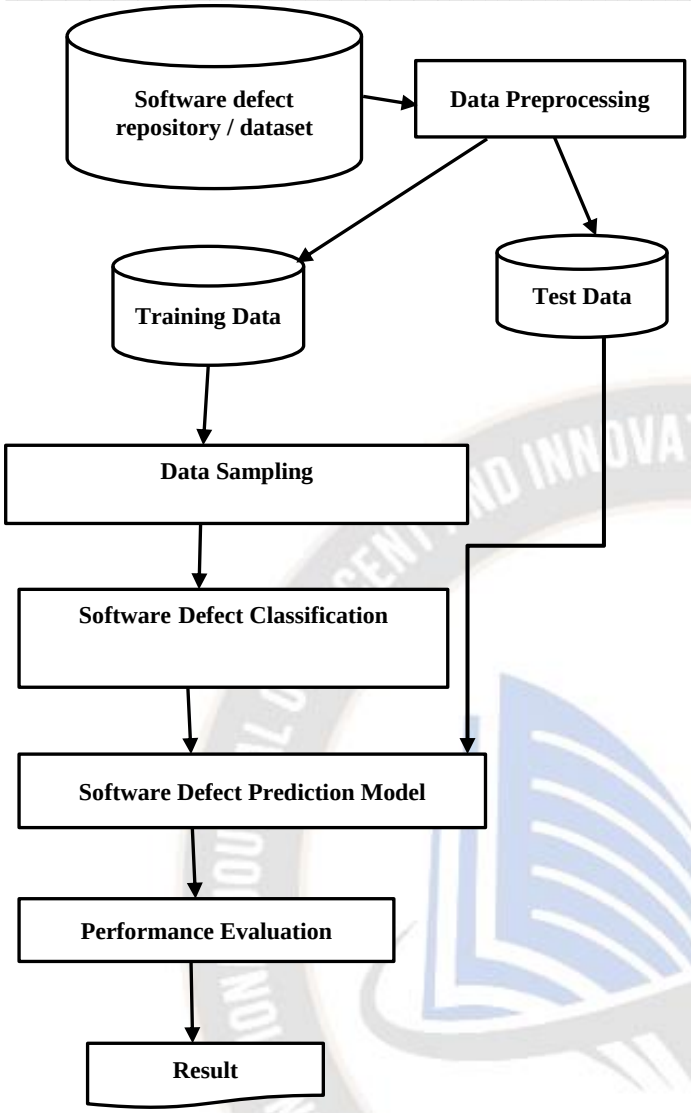

Figure 1. Software defect prediction architecture

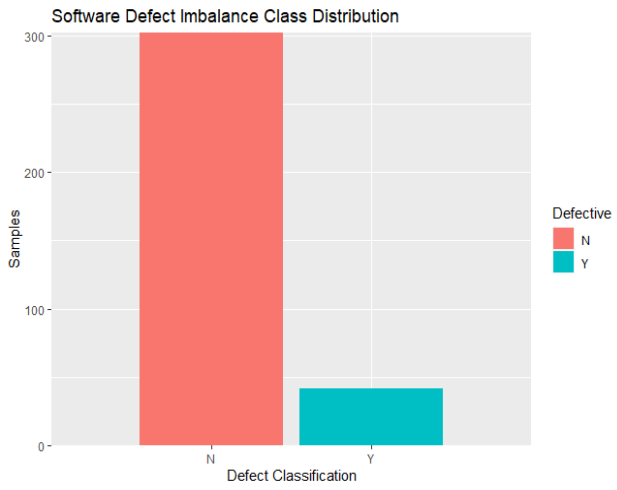

Figure 2. CM1 Software defect Imbalance class distribution

TABLE 1 GINI IMPURITY FOR CM1 DATASET

Gini impurity Value	target feature
0.193624071	locComments
0.194694632	locExecutable
0.196201667	locTotal
0.196653312	numUniqueOperators
0.198222519	numOfLines
0.198839768	numUniqueOperands
0.200464139	halsteadContent
0.202190654	numOperator
0.202424683	halsteadEffort
0.202424683	halsteadProgTime
0.202947539	halsteadVolumn
0.202957631	loc_blank
0.202963698	halsteadEffortEst
0.203642364	halsteadLength
0.203981144	numOperands
0.204238523	normCyclomaticComplex
0.204708153	designComplexity
0.204717195	percentComments
0.205095759	callPairs
0.205820916	parameterCount
0.206163297	cyclomaticComplexity
0.206237606	nodeCount
0.20677133	branchCount
0.206898721	cyclomaticDensity
0.20690006	edgeCount
0.207768467	halsteadDifficulty
0.208131643	halsteadLevel
0.208324693	decisionCount
0.208757045	modifiedConditionCount
0.209048378	ConditionCount
0.209048378	multipleConditionCount
0.209154561	maintenanceServirity
0.209526095	locCode&Comment
0.209878636	essentialComplexity
0.209920337	designDensity
0.212311526	decisionDensity
0.213018819	essentialDensity

For splitting data in CM1 dataset we have calculated entropy using following mathematical equation,

$$\text{Entropy} = -\sum_{k=1}^{N} p_k \log_2 p_k \quad \ldots\ldots\ldots\ldots (2)$$

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7461
***Article Received**: 20 May 2023 **Revised:** 15 July 2023 **Accepted:** 02 August 2023*

In equation 2 The probability of selecting a class k item at an arbitrary rate is expressed by the symbol pk and N represents number of classes and to decide the ordering of attributes in a decision tree we have calculated information gain by using following mathematical equation,

$$\text{Information Gain} = E_{Parent} - E_{Child}$$

Where, E_{parent} is the entropy of the parent node and E_{child} is the average entropy of the child nodes. Table 2 shows information gain calculated for CM1 dataset.

TABLE 2 INFORMATION GAIN FOR CM1 DATASET

Info Gain	target feature
0.273033042	decisionDensity
0.382419785	essentialDensity
0.385127871	essentialComplexity
0.461587796	modifiedConditionCount
0.462375873	parameterCount
0.468614218	branchCount
0.468614218	cyclomaticComplexity
0.469167803	decisionCount
0.470223841	multipleConditionCount
0.470464467	locCode&Comment
0.471378419	ConditionCount
0.472490352	edgeCount
0.47742667	maintenanceServirity
0.477792474	halsteadLevel
0.478567621	designDensity
0.479964826	nodeCount
0.481763928	halsteadDifficulty
0.483094208	callPairs
0.484931519	loc_blank
0.485596819	percentComments
0.48883191	designComplexity
0.490646312	cyclomaticDensity
0.491404677	numOperands
0.495544374	normCyclomaticComplex
0.49635303	locTotal
0.496574295	locExecutable
0.497813671	numOperator
0.49958927	halsteadContent
0.501045814	halsteadLength
0.501538386	halsteadEffort
0.501538386	halsteadProgTime
0.50331986	halsteadEffortEst
0.505678967	halsteadVolumn
0.519858894	numOfLines
0.519909257	numUniqueOperators
0.522310168	numUniqueOperands
0.53687742	locComments

Data Sampling:

Solving class imbalance problem in a dataset is very difficult, this can be handled by data sampling technique. Sampling will transform training dataset in balanced class distribution. For our experiment we will use Synthetic Minority Oversampling Technique which leads to improve model performance by adding minority sample in SDP (Software defect prediction) data. SMOTE Oversampling generates synthetic samples by linearly interpolating between feature vectors of neighbouring minority class samples and their close neighbours. Working procedure for Synthetic Minority Oversampling Technique will be as follows.

A new synthetic sample in feature space is created by adding the difference between a sample and its nearest neighbour, multiplied by a random value between 0 and 1, and then moving on to the next nearest neighbour for generation of new synthetic sample. Mathematically it can be expressed by,

SDPNewSample=SDPDataSample+Random(0,1)*(NeighbourSample- SDPDataSample)

Where, SDPDataSample represents SDP (Software defect prediction) minority class sample.

Random represents random value between 0 and 1 that controls the degree of interpolation.

NeighbourSample represents close neighbour of the SDPDataSample.

After applying data sampling on CM1 dataset we have balanced data as shown in figure 3

Figure 3. CM1 Software defect balanced class distribution

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7461
***Article Received**: 20 May 2023 **Revised:** 15 July 2023 **Accepted:** 02 August 2023*

Building SDP Model:

As single classifier will create bias-variance problem, due to this in our experiment we are using ensemble approach with random forest machine learning algorithm to predict software defect. Random Forest's ensemble nature helps to minimize overfitting and limiting the influence of noisy or irrelevant features in the dataset.

Our Software defect prediction model gives 81% accuracy by deploying random forest ensemble classifiers with balanced dataset.

Performance measures:

We evaluated the performance of our model using model accuracy, recall, and precision values as well as an F-measure created from the recall and precision values. Confusion matrix evaluate following parameters for measuring predictive performance of the model.

TP (True Positive): project faulty category as faulty

FP (False Positive): project non-faulty category as faulty

FN (False Negative): project faulty category as no-faulty

TN (True Negative): project non-faulty category as non-faulty

From above parameter we can calculate different model performance measure like model accuracy, recall, precision, F-measure and mathematically it can be represented as,

$$Accuracy(A) = \frac{True_{Positive} + True_{Negative}}{True_{Positive} + False_{Positive} + True_{Negative} + False_{Negative}}$$

$$Precision(P) = \frac{True_{Positive}}{True_{Positive} + False_{Positive}}$$

$$Recall(R) = \frac{True_{Positive}}{True_{Positive} + False_{Negative}}$$

$$F\_Measure(F) = \frac{2 \times P \times R}{P + R}$$

Also, for evaluating the effectiveness of our software defect categorization approach, we use a receiver operating characteristic curve by generating true versus false positives plot.

As precision-recall curve is much suitable measure for imbalanced data, so we have used precision-recall curve to assess the performance of our classification model.

IV. RESULT DISCUSSION:

In our experimental model we got performance measure result as shown in table 3.

TABLE 3 DEFECT PREDICTION PERFORMANCE MEASURE

Accuracy	Precision	Recall	F_measure
0.8142	0.8571	0.7272	0.7868

In our plot generated from our result is shown in figure 4. It shows the ROC curve is closer to the top left corner of our plot diagram, that shows our model is better in categorizing the data. To quantify this, we computed the AUC (area under the curve), which shows how much of the plot lies behind the curve. For our experimental model we got AUC (area under the curve) computed value as 0.91 and our model got highest AUC, indicating that it has the most area under the curve and is the best model at correctly categorizing observations.

The precision-recall curve shows the trade-off between Precision and Recall scores across different thresholds, the lower the threshold model will get more False Positive predictions. As shown in figure 5, the graph is trending downward and from graph we conclude that our model is good classifier.

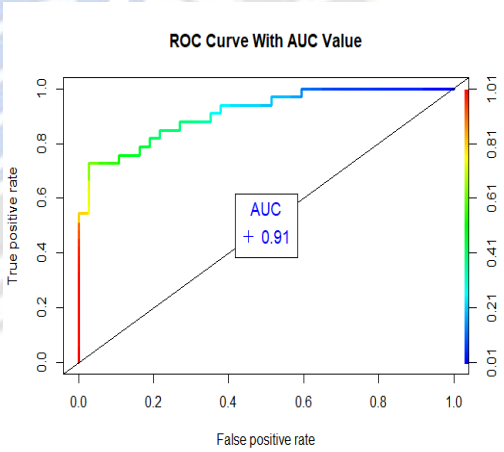

Figure 4 ROC curve for Software defect prediction

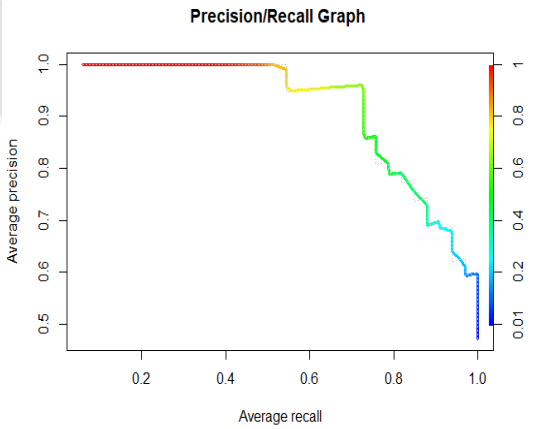

Figure 5 precision-recall curve for Software defect prediction

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7461
***Article Received**: 20 May 2023 **Revised:** 15 July 2023 **Accepted:** 02 August 2023*

CONCLUSION

In the software projects, imbalanced software defect dataset gives inaccurate classification as an outcome for software fault detection model and a bias in favour of the dominant class. Here we have first addressed high dimensional data by calculating gini impurity, entropy, and information gain for selection of important features. Then we applied data sampling to handle class imbalance in defect dataset. Finally, we build model using random forest algorithm which gives better accuracy for software defect prediction model. In future this work can be extended to cross project data for detecting defect in software product.

REFERNCES

[1] B. Pes, "Learning from high-dimensional biomedical datasets: The issue of class imbalance," IEEE Access, vol. 8, pp. 13527–13540, 2020, doi: 10.1109/ACCESS.2020.2966296.

[2] T. M. Khoshgoftaar and E. B. Allen, "Logistic regression modeling of software quality," Int. J. Reliab. Qual. Saf. Eng., vol. 6, no. 4, pp. 303–317, 1999, doi: 10.1142/S0218539399000292.

[3] R. Akbani, S. Kwek, and N. Japkowicz, "to Imbalanced Datasets," Eur. Conf. Mach. Learn., pp. 39–50, 2004.

[4] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," ACM SIGKDD Explor. Newsl., vol. 6, no. 1, pp. 20–29, 2004, doi: 10.1145/1007730.1007735.

[5] L. Breiman, "Random Forests," Mach. Learn. 45, 5-32., pp. 542–545, 2001, doi: 10.1109/ICCECE51280.2021.9342376.

[6] L. Breiman, "Bagging predictors," Mach. Learn., vol. 24, no. 2, pp. 123–140, 1996, doi: 10.1007/bf00058655.

[7] J. Ha, M. Kambe, and J. Pe, Data Mining: Concepts and Techniques. 2011. doi: 10.1016/C2009-0-61819-5.

[8] H. Krasner, "Quality Software A 2018 Report," Consort. IT Softw. Qual., 2018.

[9] R. S. Suryawanshi, A. Kadam, and D. R. Anekar, "Software defect prediction: A survey with machine learning approach," Int. J. Adv. Sci. Technol., vol. 29, no. 5, pp. 330–335, 2020.

[10] F. Akiyama, "An Example of Software System Debugging.," Int. Fed. Inf. Process. Congr., vol. 71, pp. 353–359, 1971, Accessed: Mar. 03, 2020. [Online]. Available: https://dblp.org/rec/conf/ifip/Akiyama71

[11] A. Ihara et al., "An investigation on software bug-fix prediction for open source software projects - A case study on the eclipse project," Proc. - Asia-Pacific Softw. Eng. Conf. APSEC, vol. 2, pp. 112–119, 2012, doi: 10.1109/APSEC.2012.86.

[12] Q. Yu, J. Qian, S. Jiang, Z. Wu, and G. Zhang, "An Empirical Study on the Effectiveness of Feature Selection for Cross-Project Defect Prediction," IEEE Access, vol. 7, pp. 35710–35718, 2019, doi: 10.1109/ACCESS.2019.2895614.

[13] N. K. Nagwani and S. Verma, "Predicting expert developers for newly reported bugs using frequent terms similarities of bug attributes," Int. Conf. ICT Knowl. Eng., pp. 113–117, 2011, doi: 10.1109/ICTKE.2012.6152388.

[14] S. S. Rathore and A. Gupta, "Investigating object-oriented design metrics to predict fault-proneness of software modules," 2012 CSI 6th Int. Conf. Softw. Eng. CONSEG 2012, 2012, doi: 10.1109/CONSEG.2012.6349484.

[15] P. Anand, "An approach for feature-level bug prediction using test cases," 2015 Int. Conf. Adv. Comput. Commun. Informatics, ICACCI 2015, pp. 1111–1117, 2015, doi: 10.1109/ICACCI.2015.7275759.

[16] S. Puranik, P. Deshpande, and K. Chandrasekaran, "A Novel Machine Learning Approach for Bug Prediction," Procedia Comput. Sci., vol. 93, no. September, pp. 924–930, 2016, doi: 10.1016/j.procs.2016.07.271.

[17] S. S. Rathore and S. Kumar, "Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems," Knowledge-Based Syst., vol. 119, pp. 232–256, 2017, doi: 10.1016/j.knosys.2016.12.017.

[18] J. Zheng, X. Wang, D. Wei, B. Chen, and Y. Shao, "A Novel Imbalanced Ensemble Learning in Software Defect Predication," IEEE Access, vol. 9, pp. 86855–86868, 2021, doi: 10.1109/ACCESS.2021.3072682.

[19] R. Malhotra and K. Lata, "Improving Software Maintainability Predictions using Data Oversampling and Hybridized Techniques," pp. 1–7, 2020, doi: 10.1109/cec48606.2020.9185809.

[20] Farhad Khoshbakht, Atena Shiranzaei, S. M. K. Quadri. (2023). Design & Develop: Data Warehouse & Data Mart for Business Organization. International Journal of Intelligent Systems and Applications in Engineering, 11(3s), 260–265. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/2682

[21] S. S. Rathore and S. Kumar, "An approach for the prediction of number of software faults based on the dynamic selection of learning techniques," IEEE Trans. Reliab., vol. 68, no. 1, pp. 216–236, 2019, doi: 10.1109/TR.2018.2864206.

[22] Prof. Barry Wiling. (2017). Monitoring of Sona Massori Paddy Crop and its Pests Using Image Processing. International Journal of New Practices in Management and Engineering, 6(02), 01 - 06. https://doi.org/10.17762/ijnpme.v6i02.54

[23] T. T. Khuat and M. H. Le, "Evaluation of Sampling-Based Ensembles of Classifiers on Imbalanced Data for Software Defect Prediction Problems," SN Computer Science, vol. 1, no. 2. 2020. doi: 10.1007/s42979-020-0119-4.

[24] S. Goyal, "Handling Class-Imbalance with KNN (Neighbourhood) Under-Sampling for Software Defect Prediction," Artificial Intelligence Review, vol. 55, no. 3. pp. 2023–2064, 2022. doi: 10.1007/s10462-021-10044-w.

[25] M. Gan, Z. Yücel, and A. Monden, "Neg/pos-Normalized Accuracy Measures for Software Defect Prediction," IEEE Access, vol. 10, no. November, pp. 134580–134591, 2022, doi: 10.1109/ACCESS.2022.3232144.

[26] P. Soltanzadeh and M. Hashemzadeh, "RCSMOTE: Range-Controlled synthetic minority over-sampling technique for handling the class imbalance problem," Inf. Sci. (Ny)., vol. 542, pp. 92–111, 2021, doi: 10.1016/j.ins.2020.07.014.

[27] F. Thabtah, S. Hammoud, F. Kamalov, and A. Gonsalves, "Data imbalance in classification: Experimental evaluation," Inf. Sci. (Ny)., vol. 513, pp. 429–441, 2020, doi: 10.1016/j.ins.2019.11.004.

[28] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7461
***Article Received**: 20 May 2023 **Revised:** 15 July 2023 **Accepted:** 02 August 2023*

comments on the NASA software defect datasets," IEEE Trans.
Softw. Eng., vol. 39, no. 9, pp. 1208–1215, 2013.