IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

PREDICTING A FAULTY PARAMETER USING MACHINE LEARNING - AN ANALYSIS

Dimple Bisht[1],Dipti[2],Gaurav Kotecha[3],Meghna Jha[4]

Rekha Jayaram[5]

1,2,3,4 Students, Department of Information Science and Engineering, Dayananda Sagar College of Engineering Bangalore, Karnataka, India

5 Associate Professor, Department of Information Science and Engineering, Dayananda Sagar College of Engineering Bangalore, Karnataka,India

Abstract: Today's software systems give us a wealth of information about both the points where the system performs as expected and the points where it fails.

The knowledge base which can be established with the help of this data will help us in delivering an insight for any future act or manner of performance for the specified software.

This research aims at looking for machine learning algorithms which can aid us in anticipating the behaviour of the software in consideration and hence also give an idea about the spots where the output of the system is not as desired,.i.e a defective parameter.

This paper mainly looks over the creation of a knowledge base with the previously tested data and also predicts the areas and data values where a user can expect the system to be showing erroneous behavior.

Index Terms **- software fault prediction, knowledge based system, correlation based feature selection, prediction models, supervised machine learning.**

I.INTRODUCTION

"Fault" with respect to a software is a scenario where there is a malfunction in the dynamic system. This further ends up causing an anomaly in the overall software functionality.Both the user experience and system performance may be significantly impacted by this. Every piece of commercially available software must undergo a rigorous testing process, which can be time- and money-consuming for the developers.

The output of a set of parameters can help us identify and predict the areas where a system might deviate from the expected behavior, which has led to a boom in machine learning research.There are many machine learning techniques which can be used for this but we have to always keep the final prediction accuracy in mind before going ahead and implementing in the final model.In this paper,our focus lies mainly on machine learning using knowledge based systems for 'prediction' of parameters which 'can' be faulty.

Knowledge base system can be thought of as a computer system which keeps a database which is progressively developed by each run of the software.It can be stated to have a history of all the instances in which data was sent to software as an input and what results were as a result.

Using the knowledge base created by a few genesis runs of the software, we can further train an appropriate algorithm.

Our goal now becomes to use this knowledge base and our machine learning model to let us know if our given set of input values will end up showing an error in the software,without actually running the software.

This ,if realized ,can lead to great cost,time and energy savings for the software testers and also enhance the end user experience.

II. RELATED WORK:

Here,we cover five papers already published for the study of faulty parameters prediction.

The first paper[5] works around answering two main research questions.The first i**s does the manipulation of parameters lead to introduction of any performance enhancement** with respect to measures of **accuracy**.The other one is that amongst the algorithms studies**,which provides a better SFP performance.**Here a number of experiments were conducted and evaluated using metrics of accuracy,detection rate,TNR.Here,the results from CNN showed high accuracy.The conclusion was also that the increase

in number of layers has an overall positive effect on the optimal layers in each data set.For activation functions,ReLU showed highest performance.
The final conclusion,**hence,was that the impact of enhancement of parameters led to exceptional effect and gave good results.**
The main limitation here was not investigating the effectiveness of all hyper parameters and the authors intend to address this in their future work.The future scope also includes more experiments and utilizing other data sets to dive deeper and check whether the data set actually plays an important role or the final result depends on the accuracy of the chosen algorithm only.
Hence,the aim of this study is to find the best deep learning algorithms used for Software Defect Prediction(SFP).Also,the it would be worthwhile to investigate the relationship between dataset and its approximate fault ratio with the selected algorithm and parameters.Finally ,we need to develop a tool that uses a deep learning algorithms and further for other fields.

The idea primarily states that while we develop a knowledge base, the data giving correct or non-faulty outputs is much more in comparison to the data with the faulty outputs. This creates an imbalance due to which many machine learning algorithms fail. Hence, the concept of class imbalance learning is introduced. The authors here investigate various types of class imbalance learning.
Static properties are taken from past software releases, together with the defect log file, and these are then utilized to generate a prediction to pinpoint the problematic modules in the upcoming release, i.e. identifying the area of the software with the highest flaws.
McCabes and Halstead metrics are taken into account for the software's performance, and the learning algorithms investigated here include Naive Bayes, Random Forest, and AdaBoost.
By taking a flow graph into account, the McCabes metrics inform us of the path complexity in a module.
Based on the number of operators and operands in the specified module, Halstead metrics are used to estimate the reading difficulty.
However, none of these techniques take into account the distribution's extreme imbalance between classes with defects and those without, which eventually results in the produced model's subpar overall performance.
The authors next carry out an organized analysis of 5 classes and 2 high performance predictors.They make use of 10 open data sets from real-world software projects that were gathered with the help of the PROMISE repository.
(Promise repository is a collection of defect prediction data sets that are donated to by the real world software projects which are in public use.)

The attention that needs to be placed on the minority classes is then identified by using the parameter searching strategy to these teaching techniques.
The main goal therefore shifts to finding or creating a classifier that will give us high accuracy for the minority class while also maintaining accuracy for the majority class.
To correct skewed distributions, data-level techniques like resampling and training data manipulation are used.
The training techniques are also changed at the algorithmic level.
Additionally, the ensemble learning approach is presented, which combines the strengths of individual students to improve performance as a whole.

Below is a diagram of the experimental study's conceptual framework:

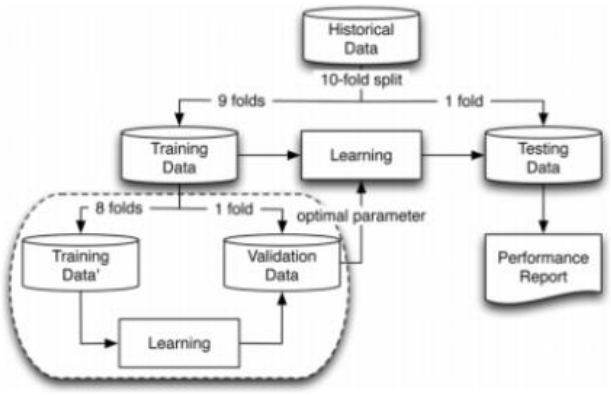

fig1: framework of the experimental studies

The Naive Bayes model with log filters and random forest regressors is then compared to these results.
The study's findings reveal that when the characteristics of balance, Geometric Means (Gmeans), and area under the curve are taken into account, the AdaBoost.NC algorithm exhibits the best overall performance.
The best method for defect detection is the Naive Bayes.

The idea of integrating AdaBoost.NC and Naive Bayes approaches to automatically alter the parameter during model training based on performance criteria is presented as the paper's conclusion.
Although class imbalance is one of the major issues in defect detection, this study discusses it without offering any firm recommendations for solutions.

Future research from this publication should look into the various base classifiers, as only C4.5 decision trees were taken into consideration.
We also need to take a closer look at additional real-world examples of Software Defect Prediction (SDP), which involves gathering information from programs with few problematic modules, lots of unsupervised or unlabeled modules, and the isolation of faults to identify their specific nature.

In the third paper[2,] researchers analyzed different machine learning algorithms based on their accuracy, recall, precision, and f-measure in order to detect system or software faults.They have employed both correlation-based feature selection (CFS) and principal component analysis (PCA) to choose the features from the original input set.
began with the use of independent software code measurements for determining whether the object is defective or not, followed by prediction.The dataset is extracted from the sizable Promise repository, part of the NASA project.The entire implementation of the system was carried out in Java, and it is an open source system.
Based on the average of the 10 tested classifiers, performance is evaluated.

Prior to employing feature selection approaches, the results were obtained using Naive Bayes, which had the lowest accuracy, and Random Forest or an ANN (artificial neural network), which had produced the best results. Second, employing feature selection techniques include CFS (correlation-based feature selection), PCA (principal component analysis), rankers, and greedystepwise

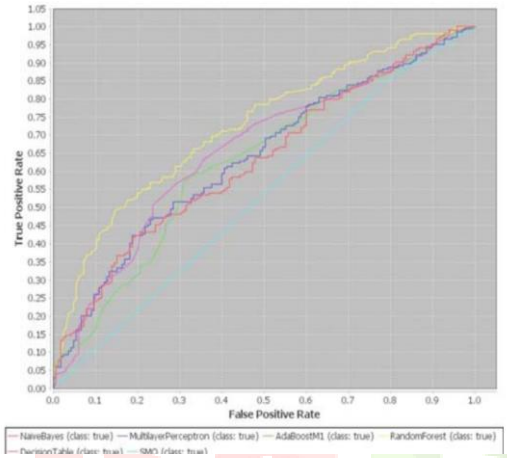

fig2: ROC curves of the classifiers when PCA is applied

Rankers is the method of selecting the attributes based on their single evaluation as classifiers whereas greedystepwise follows the backward and forward approach for searching or selecting the attributes.

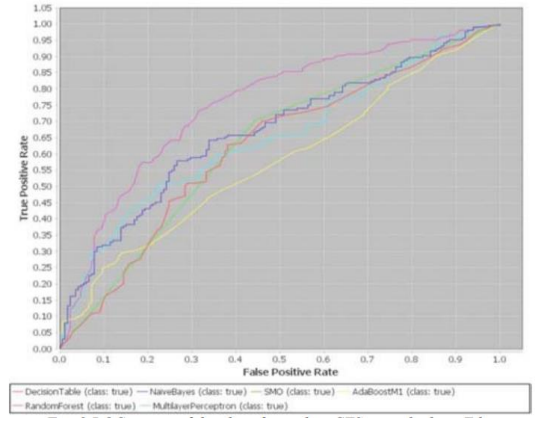

fig3:ROC curves of the classifiers when CFS is applied

To better understand the results, the **ROC (Receiver operating Curve**) curve was plotted for both forms of selection. When compared to other approaches, such as SVM, the graphs for selection methods without characteristics demonstrate that **Random forest has improved accuracy** and AUC(area under curve) value the greatest AUC value increases abruptly from **0-9%** after feature selection using PCA and CFS.This means that **SVM couldn't perform with the multiple-dimensional characteristics whereas Random forest did well with the CFS.**
This indicates that for given little to large datasets, CFS is superior to PCA.

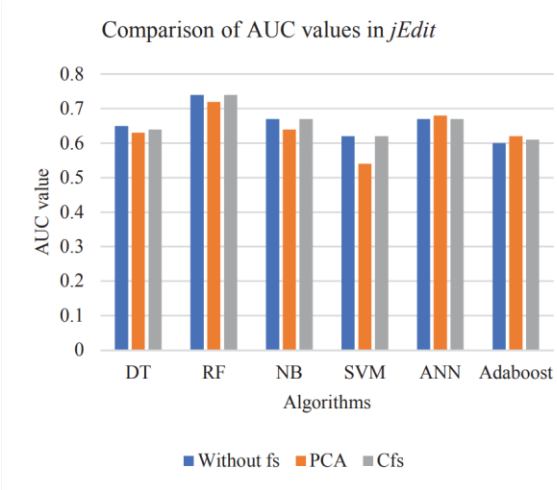

Fig4: AUC value comparison

The fourth paper[3] discusses the usage of knowledge based systems for the prediction of faults and verification and validation of the software applications which is the major step for any fault prediction application. The main focus has been given on the steps for the **development of knowledge based systems** and the difficulties faced while testing these systems.The different steps used are : the testing of the **criteria** used for the software application, the **generation** of inputs and outputs and then fault prediction.

General Framework for V&V
1) Determination of goals of V&V
2) Generation of problems (inputs) to be used by a V&V method
3) Generation of expected outcomes (outputs) on the selected problems
4) Application and Evaluation of problems

Fig 6:V&V Framework

The first step is to consider the different knowledge bases (KBs) which may include rules, logics and networks like parameter combinations can be taken into consideration to arrive at a set of parameters that can most effectively give the fault predictions. The paper also says that the terminologies being used should be standardized and formal semantics should be established. Object oriented or semantics on which the application is developed, often called the knowledge representation formalisms. As the versions of these applications change, a dynamic knowledge based system may be useful. Firstly, the criteria is decided which will lead to the parameters we are considering for the machine learning algorithms to work on. The input data that best matches the test criteria is then selected.

To better understand the results, the ROC (Receiver operating Curve) curve was plotted for both forms of selection. When compared to other approaches, such as SVM, the graphs for selection methods without characteristics demonstrate that Random forest has improved accuracy and AUC value the greatest. AUC value increases abruptly from 0-9% after feature selection using PCA and CFS.This means that SVM couldn't perform with the multiple-dimensional characteristics whereas Random forest did well with the CFS.
This indicates that for given little to large datasets, CFS is superior to PCA.

Input is **verified, and test data** is collected into tables where error-detection algorithms are conducted. You may carry out a lot of experiments and different jobs with the application.
For four sets of real-time software defect data, the final paper[4] discussed here assesses the various predictive models that are now available. Real-time software systems, according to the documentation, need to act dynamically in order to adjust to particular operating conditions and runtime requirements. As a result, a real-time evaluation mechanism must be created to categorize dynamically formed systems as flawed or fail-safe. **Dynamic stability has the advantage** of giving operators feedback so they can alter mission objectives.

Numerous methods for predicting software defects have been put forth, but **none have been shown to be consistently reliable.**There are a variety of statistical methods, machine learning techniques, and parametric models, but it is still necessary to identify the top prediction method for a particular issue.Numerous statistical techniques are employed, but progress has also been made in the study of machine learning strategies for software quality assessments.Simulation was utilized to examine rule induction, case-based reasoning (CBR), artificial neural networks (ANN), and stepwise regression in software prediction**.Stepwise regression performed better when the goal function was continuous, but other methods performed better when the target function was discontinuous.**

Since it turned out to be **better by a slim margin, CBR did gain favor**.Narrowly focusing on **accuracy, neural networks performed better**.Additionally, research was conducted to look into ways to merge statistical and machine learning techniques.
There was an increase in overall performance of neural networks when PCA(Principal Component Analysis)was used.
However, it was discovered that **feature subset selection (FSS) methods outperform PCA.**
Then it was suggested that these prediction models, with all of their benefits and drawbacks, could result in risk management choices that turned out to be incorrect because they failed to adequately account for the interconnections between qualities
.
Bayesian Belief Networks (BNN) were therefore suggested as an alternative, but building the topology of a BNN turns out to be rather a difficult undertaking.Then it was claimed that by looking into software decompositions that successfully capture these informal links, a better science of software engineering may be attained.
This study also demonstrates how several elements, such as multicollinearity among variables and the prevalence of defects in existing data sets, have an impact on software defect prediction models.
PCA was used to eliminate multicollinearity, however in most cases there was no discernible reduction in the error prediction of learning models.
Additionally, **no connection between the accuracy and skewness** of machine learning techniques was discovered. IBL must unquestionably be taken into consideration because CFS and CBS performed better in the majority of cases, and especially for IBL approach.

Furthermore, the **random attribute reduction did not result in better accuracy,** indicating that the process of attribute reduction should be carried out in a certain way.(like in CBS and CFS).
The researchers r**ecommend 1R and instance-based learning techniques**, but they issue a warning that there is no clear pattern in the accuracy forecast of various techniques and data sets.
Picking the best result set from a group of prediction models might be the best course of action.

In order to uncover flawed real-time software models, this article compares various machine learning methods.The **finding is that no single learning method outperforms** another for all data sets.**IBL and 1R are more reliable** approaches that consistently make accurate predictions.
There were **no computational benefits added by the attempt to combine machine learning techniques** with **statistical methods like PCS**.In contrast, straightforward **FSS techniques outperformed PCA**.
Additionally, it was determined that "**size" and "complexity" measures are insufficient as criteria** for precise prediction.To increase forecast accuracy, it is necessary to take into account the **interconnections between these metrics**.To comprehend these ad hoc relationships, one can investigate the **Bayesian Belief Network** approaches.
Future work will focus on creating an analytical strategy to better explain the analysis results and look into Bayesian Belief Networks.

III.CONCLUSION:

Here, we have examined five works on fault prediction in an effort to learn more about it.
The main study deals with determining which algorithm would produce a decent result and then questions connected to parameter modification in software prediction.In the paper's conclusion, parameter combinations in software prediction models are supported.
The second paper's expertise demonstrates a focus on data imbalance.To achieve a specialization, class imbalance techniques like AdaBoost, NC are utilized.
We were able to examine several machine learning algorithms thanks to the second paper, we came to the conclusion that Random forest and Naive Bayes performed the best without feature selection.This demonstrates that Random Forest and Correlation Based Feature Selection produced the most accurate results.
The third paper provides parameters that are similar to rules, logics, and networks that can be used to create a final set of parameters for successful fault prediction.
Finally, we investigate an empirical evaluation of prediction accuracy, defect count, and statistical prediction systems utilizing WEKA software. It demonstrates that a discrete attribute's projected fault is lower than a continuous attribute. Results indicate that IBL and 1R are the superior approaches. Additionally, it demonstrates that utilizing PCA may not be wise in cases with software flaws.We also shown that measures for "size" and "complexity" are insufficient for reliable prediction. To make the prediction models more accurate, we must take into account the dependencies between these indicators in our analysis.

Therefore, this acquired information will assist us in identifying additional rules and logics to choose parameters, identify input values with dependencies, if any, and finally, apply techniques like CFS and Random Forest to construct a model for fault prediction.

IV.REFERENCES:

[1]:Shuo Wang, Member, IEEE, and Xin Yao, Fellow, IEEE- "Using Class Imbalance Learning for Software Defect Prediction".

[2]: Guru Prasad Bhandari DST-Centre for Interdisciplinary Mathematical Sciences Institute of Science, Banaras Hindu University,Ratneshwar Gupta School of Computer & Systems Sciences Jawaharlal Nehru University -"Machine learning based software fault prediction utilizing source code metrics"

[3]: Wei-Tek Tsai, Member, IEEE Computer Society, Rama Vishnuvajjala, and Du Zhang, Member, IEEE- "Verification and Validation of Knowledge-Based Systems".

[4]:Venkata U.B. Challagulla, Farokh B. Bastani, I-Ling Yen-Department of Computer Science University of Texas at Dallas, Raymond Paul Department of Defense- "Empirical Assessment of Machine Learning based Software Defect Prediction Techniques ".

[5]:Osama Al Qasem,Mohammaed Akour and Mamdouh alenzi-"The influence of deep learning algorithm factors in software fault prediction",March 2020

[6]:N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "SMOTE-Boost: Improving prediction of the minority class in boosting," Knowl- edge Discovery in Databases: PKDD 2003, vol. 2838, pp. 107–119

[7]:A. Avižienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," IEEE Trans. Dependable Secur. Comput., vol. 1, no. 1, pp. 11–33, https://doi.org/10.1109/TDSC.2004.2.

[8]:M. Ayel and J.P. Laurent, "SACCO-SYCO JET: Two Different Ways of Verifying Knowledge-Based Systems," Validation, Verification, and Test of Knowledge-Based Systems, M. Ayel and J.P. Laurent, eds., John Wiley and Sons, Chichester, U.K.

[9]:Schneidewind, N.F., "Investigation of logistic regression as a discriminant of software quality", Software Metrics Symposium, METRICS 2001. Proceedings. Seventh International, Page(s): 328 -337.

[10]:] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, ''Dropout: A simple way to prevent neural networks from overfitting,'' J. Mach. Learn. Res., vol. 15, no. 1, pp. 1929–1958, 2014.

[11]:D. Ghosh and J. Singh, ''A novel approach of software fault prediction using deep learning techniques,'' in Automated Software Engineering: A Deep Learning-Based Approach. Cham, Switzerland: Springer, 2020.