Identify the bugs from Software using the Supervised algorithms

Ms. Rupinder Kaur
Research Student, Dept. of ECE, Bhai Gurdas Institute of Engineering and Technology – Affiliated with Maharaja Ranjit Singh Punjab Technical University, Punjab, India,

Mr. Rishav Dewan
Assistant Professor, Dept. of ECE, Bhai Gurdas Institute of Engineering & Technology – Affiliated with Maharaja Ranjit Singh Punjab Technical University, Punjab, India.

Abstract: Software Bug Detection (SBD) is the critical concern in the field of the software development process. Because if we can identify the errors in the software in the initial phase that will help to enhance the quality and efficiency of the software. But to implement such software is the main challenge and that's why we need to analyze multiple machine learning algorithms from time to time. Multiple classifiers includes Decision Tree, Naive Bayes, KNN and many other relevant algorithms. Thus, we need to check the accuracy obtained and need to consider the most accurate one in consideration. The results, we observed will show the most accurate results of the prediction.

Index Terms – Software bug Detection; ML algorithms; Decision Tree; KNN, Naive Bayes, Artificial Neural Networks (ANNs)

I. INTRODUCTION

Software engineering is concerned with the development of software products. Properly engineered systems are reliable and they satisfy user requirements while at restricted the imperative control flow to hierarchical structures instead of ad-hoc jumps. Computer programs written in this style were more readable, easier to understand, and reason about. Another improvement was the introduction of an object-oriented paradigm [1] as a formal programming concept. In the early days, software engineers perceived significant similarities between software and civil engineering processes. The waterfall model [2], which was widely accepted as such regardless of its original description, actually recommending the agile methodology.

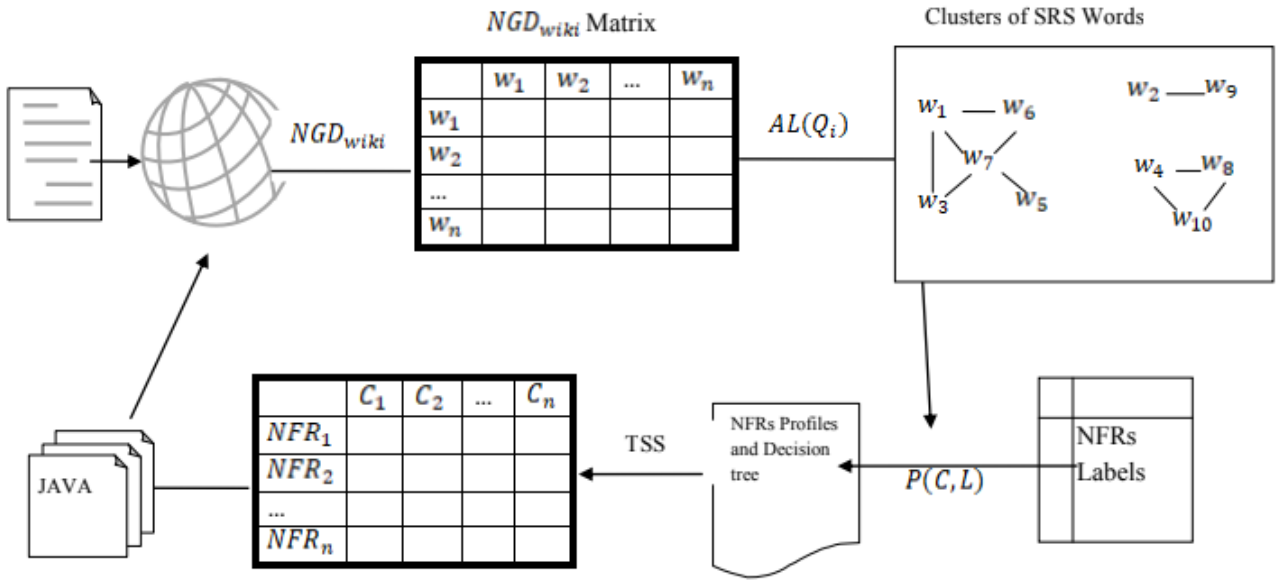
It has soon turned out that building software differs from building skyscrapers and bridges, and the idea of extreme programming emerged [3], its key points being: keeping the code simple, review it frequently, and early and frequent testing. Among numerous techniques, a test-driven development was promoted which eventually resulted in the increased quality of produced software and the stability of the development process [4]. Contemporary development teams started to lean towards short iterations (sprints) rather than fragile upfront designs, and short feedback loops, thus allowing customers' opinions to provide timely influence on software development. This meant creating even more complex software systems. The growing complexity of software resulted in the need to describe it at different levels of abstraction, and, in addition to this, the notion of software architecture has developed. The emergence of patterns and frameworks had a similar influence on the architecture as design patterns and idioms had on programming. Software started to be developed by assembling reusable software components which interact using well-defined interfaces, while component-oriented frameworks and models provided tools and languages making them suitable for formal architecture design.

II. OBJECTIVES

1. Implement textural semantic techniques for the classification of non-functional requirements.
2. To propose improvement in textural semantic technique using SVM classifier for the classification of non-functional requirements.
3. Implement the proposed technique for the classification of non-functional requirements.
4. Analyze the results of the proposed technique in terms of accuracy, execution time.

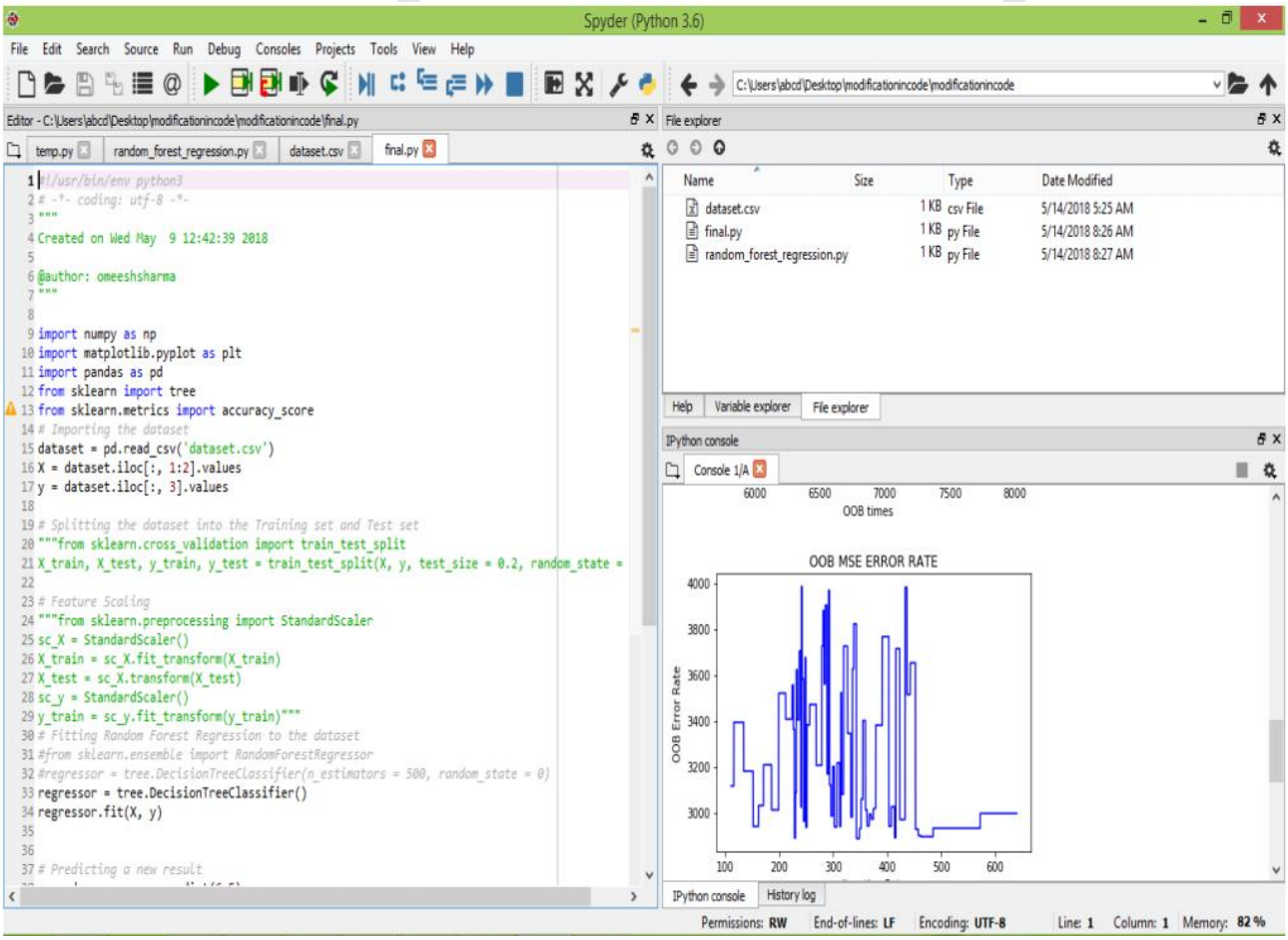
III. RESEARCH METHODOLOGY

The potential NFRs present within the system is identified within the initial phase of this analysis. On the basis of major assumptions of the cluster hypothesis, the research assumptions are generated within this phase. On the basis of relevance of information required [5], the documents that behave similarly are present within similar clusters. On the basis of individual words of information content present under the documents, this behavior is attached. Thus, the clustering of words within the same cluster is done specifically. On the basis of this approach, certain assumptions are used. The generation of semantically coherent groups of natural language words is the major objective of clustering the keywords that are extracted from the functional requirements of the software system. Group of words W = {w1, w2,...,Wm} into a set of disjoint groups of semantically similar words are clustered such that a C={c1, c2,... ck} is generated which is a set of disjoint groups that include semantically similar words. The conceptual themes that pass through the original text need to be described ideally by the clusters in C. the quality constraints of a system are mainly represented with the help of such themes. An NP-hard issue that arises here is the identification of optimal cluster configurations that are best Fitted. However, with the help of certain experiments, near-optimal solutions can be achieved. The performances of a series of semantic similarity measures as well as the word clustering techniques are evaluated in order to determine such configurations. Thus, from the FR words, the semantically coherent or thematic clusters or groups can be generated.



As shown in the figure, the proposed approach is explained according to the steps followed. Here, NGD stands for Normalized Google Distance, AL stands for Average Linkage clustering algorithm, the cluster quality objective function is denoted by Qi the classification formula is given by P(C, L), TSS stands for text semantic similarity as well as the code class is represented by Ci. In this research work, the decision tree classifier is applied for the classification. These are the trees in which instances are classified by arranging them on the basis of feature values. A feature of an instance is represented by each node in a decision tree for the classification assumed value by the node is represented by each branch. These instances are classified at the root node initially after which on their feature values they are sorted. This learning has been utilized in data mining and machine learning. It is a predictive model in which an item is observed on the map in order to identify the item's target value. These three models are also known as classification trees or regression trees. For the evaluation of decision trees, post-pruning techniques have been utilized as they are pruned by using a validation set. This research work is based on the classification of null and not null functions from the IVR dataset. For this, the NFR matrix is applied/combined with the decision tree classifier for the classification of data. The performance of the proposed and existing method is compared in terms of accuracy and execution time. In the existing/old method only NFR matrix is applied for the classification of data. In the proposed method, the decision tree is applied along with the NFR matrix for the data classification.

IV Comparison of Existing and Proposed Model



As shown in figure, the IVR dataset is loaded into training and test set for the classification. The model is build for the classification using training and test set. The decision tree classifier is applied for the null functional requirements. The accuracy of the classification model is approximately 79 percent

V. CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

It has been concluded that reverse engineering is an efficient technique to generate source code. The process of extraction, abstraction, and visualization is used for the generation of source code. The classification technique used is Decision Tree Classification which provides the more accurate results and having less error rates.

5.2 Future Scope

Software defect prediction using machine learning algorithms is a very efficient and effective way to predict the defects in any software application in very less time. Further investigation can be done by different machine learning algorithm for improvement in the prediction accuracy.

VI. ACKNOWLEDGEMENT

I am thankful to the creator of this universe who created me as a human being. I am also thankful to my guide and my family members, who always support, help and guided me during my work. I specially thank to my childhood teachers who groomed me well to innovate new ideas in this world of competition. I especially thanks to my special to the unparalleled mother who is there like a sheath and shadow for me. She always supports my innovative ideas. I am thankful to my college friends who helped me a lot and at last, I am thankful to all relatives who pray for my astonishing success.

REFERENCES

[1] T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” IEEE Transactions on Software Engineering, Vol. 33, No. 1, 2007, pp. 2–13.
[2] E.W. Dijkstra, “Letters to the editor: go to statement considered harmful,” Communications of the ACM, Vol. 11, No. 3, 1968, pp. 147–148.
[3] J. McCarthy, P.W. Abrams, D.J. Edwards, T.P. Hart, and M.I. Levin, Lisp 1.5 programmer’s manual. The MIT Press, 1962.
[4] W. Royce, “Managing the development of large software systems: Concepts and techniques,” in Technical Papers of Western Electronic Show and Convention (WesCon), 1970, pp. 328–338.
[5] K. Beck, “Embracing change with extreme programming,” IEEE Computer, Vol. 32, No. 10, 1999, pp. 70–77.