International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 06 Issue: 06 | June - 2022 Impact Factor: 7.185 ISSN: 2582-3930

IDENTIFING SOFTWARE BUGS OR NOT USING SMLT MODEL

Mr. G. Rajasekaran[1], Abdur Rahman[2], Abishek M[3], Abubucker Siddique[4]

[1]*Associate Professor, Dept of AI&DS, Dhaanish Ahmed College of Engineering, Chennai.*

[2,3,4]*Final year student, Dept of CSE, Dhaanish Ahmed College of Engineering, Chennai.*

ABSTRACT-Defect classifiers are widely used by many large software corporations. Defect classifiers are commonly interpreted to uncover insights to improve software quality. Such insights help practitioners formulate strategies for effective testing, defect avoidance, and quality assurance [8, 9]. Therefore, it is pivotal that these generated insights are reliable. When interpreting classifiers, prior studies typically employ a feature importance method to compute a ranking of feature importance (a.k.a., feature importance ranks). These feature importance ranks reflect the order in which the studied features contribute to the predictive capability of the studied classifier. These feature importance methods can be divided in two categories: classifier-specific (CS) and classifier-agnostic (CA) methods. The objective is to propose an improvement to the existing study on the importance of feature ranking in terms of defect classifiers..

Key words : Identification of Software Defects or not using SMLT Model

INTRODUCTION

Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from structured and unstructured data, and apply knowledge and actionable insights from data across a broad range of application domains. The term "data science" has been traced back to 1974, when Peter Naur proposed it as an alternative name for computer science. In 1996, the International Federation of Classification Societies became the first conference to specifically feature data science as a topic. However, the definition was still in flux. The term “data science” was first coined in 2008 by D.J. Patil, and Jeff Hammerbacher, the pioneer leads of data and analytics efforts at LinkedIn and Facebook. In less than a decade, it has become one of the hottest and most trending professions in the market. Data science is the field of study that combines domain expertise, programming skills, and knowledge of mathematics and statistics to extract meaningful insights from data. Data science can be defined as a blend of mathematics, business acumen, tools, algorithms and machine learning techniques, all of which help us in finding out the hidden insights or patterns from raw data which can be of major use in the formation of big business decisions.

OBJECTIVE

The goal is the comparison of different defect classifiers while taking into note the agreeableness of the Classifier Agnostic and Classifier Specific feature importance ranking methods while improving upon the existing study.

LITERATURE SURVEY

Title : A Systematic Literature Review of Software Defect Prediction: Research Trends, Datasets, Methods and Frameworks 11 Author: Romi Satria Wahono

Year : 2015

Recent studies of software defect prediction typically produce datasets, methods and frameworks which allow software engineers to focus on development activities in terms of defect-prone code, thereby improving software quality and making better use of resources. Many

software defect prediction datasets, methods and frameworks are published disparate and complex, thus a comprehensive picture of the current state of defect prediction research that exists is missing. This literature review aims to identify and analyse the research trends, datasets, methods and frameworks used in software defect prediction research between 2000 and 201377.46% of the research studies are related to classification methods, 14.08% of the studies focused on estimation methods, and 1.41% of the studies concerned on clustering and association methods. In addition, 64.79% of the research studies used public datasets and 35.21% of the research studies used private datasets. Nineteen different methods have been applied to predict software defects. From the nineteen methods, seven most applied methods in software defect prediction are identified. Researchers proposed some techniques for improving the accuracy of machine learning classifier for software defect prediction by ensembling some machine learning methods, by using boosting algorithm, by adding feature selection and by using parameter optimization for some classifiers. The results of this research also identified three frameworks that are highly cited and therefore influential in the software defect prediction field. They are Menzies et al. Framework, Lessmann et al. Framework, and Song et al. Framework.

Title : Anomaly-Based Bug Prediction, Isolation, and Validation: An Automated Approach for Software Debugging

Author Martin Dimitrov and Huiyang Zhou Year : 2009

Software defects, commonly known as bugs, present a serious challenge for system reliability and dependability. Once a program failure is observed, the debugging activities to locate the defects are typically nontrivial and time consuming. In this paper, we propose a novel automated approach to pin-point the root-causes of software failures. Our proposed approach consists of three steps. The first step is bug prediction, which leverages the existing work on anomaly-based bug detection as exceptional behaviour during program execution has been shown to frequently point to the root cause of a software failure. The second step is bug isolation, which eliminates false-positive bug predictions by checking whether the dynamic forward slices of bug predictions lead to the observed program failure. The last step is bug validation, in which the isolated anomalies are validated by dynamically nullifying their effects and observing if the program still fails. The whole bug prediction, isolation and validation process is fully automated and can be implemented with efficient architectural support. Our experiments with 6 programs and 7 bugs, including a real bug in the gcc 2.95.2 compiler, show that our approach is highly effective at isolating only the relevant anomalies. Compared to state-of-art debugging techniques, our proposed approach pinpoints the defect locations more accurately and presents the user with a much smaller code set to analyze.

Title : An Empirical Study on the Use of Defect Prediction for Test Case Prioritization

Author: David Paterson, Jose Campos, Rui Abreu

Year : 2019

Test case prioritization has been extensively researched as a means for reducing the time taken to discover regressions in software. While many different strategies have been developed and evaluated, prior experiments have shown them to not be effective at prioritizing test suites to find real faults. This paper presents a test case prioritization strategy based on defect prediction, a technique that analyses code features – such as the number of revisions and authors — to estimate the likelihood that any given Java class will contain a bug. Intuitively, if defect prediction can accurately predict the class that is most likely to be

buggy, a tool can prioritize tests to rapidly detect the defects in that class. We investigated how to configure a defect prediction tool, called Schwa, to maximize the likelihood of an accurate prediction, surfacing the link between perfect defect prediction and test case prioritization effectiveness. Using 6 realworld Java programs containing 395 real faults, we conducted an empirical evaluation comparing this paper's strategy, called G-clef, against eight existing test case prioritization strategies. The experiments reveal that using defect prediction to prioritize test cases reduces the number of test cases required to find a fault by on average 9.48% when compared with existing coverage-based strategies, and 10.5% when compared with existing history-based strategies.

Title: Gram-Schmidt Orthogonalization for feature ranking and selection - A case study of claim prediction

Author: Yuni Rosita Dewi, Hendri Murfi, Yudi Satria

Year : 2020

Claim prediction is an important process in the insurance industry to prepare the right type of insurance policy for each potential policyholder. The frequency of claim predictions is highly increasing that head the problem of big data in terms of both the number of features and the number of policyholders. One of machine learning paradigms to handle the problem of the big data is dimensionality reduction by using a feature selection method. In this paper, we examine a new feature selection method for claim prediction using GramSchmidt Orthogonalization. In this method, the next features are iteratively selected based on the farthest distance to space spanned by the current features. Therefore, the advantage of the Gram-Schmidt Orthogonalization method is that it can provide a subset of the feature ranking without ordering all features. Our simulation shows that by using only about 26% of features, the predictor can reach comparable accuracy when it uses all features. It means that the GramSchmidt Orthogonalization-based feature selection method may need memory usage of about 26%, which is very significant in the context of the Big Data problem.

EXISTING SYSTEM:

Thought:

Insights are generated from the feature importance ranks that are computed by either CS or CA methods. However, the choice between the CS and CA methods to derive those insights remains arbitrary, even for the same classifier. In addition, the choice of the exact feature importance method is seldom justified..

Technique: CS or CA Methods

Damage: Correlation based Feature selection is not suitable if the features are mutually correlated.

PROPOSED SYSTEM:

Thought:

The usage of advanced feature interaction removal method, GramSchmidt is taken into account for the identification and removal of feature interactions in order to increase the agreeableness of CS and CA feature rankings.

Technique: Naive-Bayes Algorthim,Random Forest Algorithm

Advantage: Performance metrics of different algorithms are compared and the better prediction is done.

SYSTEM IMPLEMENTATION

ALGORITHM USED:

1.Random Forest Classifier

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by

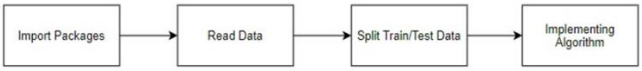
constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random forest is a type of supervised machine learning algorithm based on ensemble learning. Ensemble learning is a type of learning where you join different types of algorithms or same algorithm multiple times to form a more powerful prediction model. The random forest algorithm combines multiple algorithms of the same type i.e., multiple decision *trees*, resulting in a *forest of trees*, hence the name "Random Forest". The random forest algorithm can be used for both regression and classification tasks.

The following are the basic steps involved in performing the random forest algorithm:

- Pick N random records from the dataset.
- Build a decision tree based on these N records.
- Choose the number of trees you want in your algorithm and repeat steps 1 and 2.

In case of a regression problem, for a new record, each tree in the forest predicts a value for Y (output). The final value can be calculated by taking the average of all the values predicted by all the trees in forest. Or, in case of a classification problem, each tree in the forest predicts the category to which the new record belongs. Finally, the new record is assigned to the category that wins the majority vote.

MODULE DIAGRAM



2. Naïve Bayes Classifier Algorithm

➢ Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.

➢ It is mainly used in text classification that includes a high-dimensional training dataset.

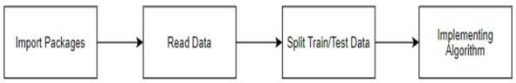
➢ Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.

➢ It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

➢ Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

This process is continued on the training set until meeting a termination condition. It is constructed in a top-down recursive divide-and-conquer manner. All the attributes should be categorical. Otherwise, they should be discretized in advance. Attributes in the top of the tree have more impact towards in the classification and they are identified using the information gain concept. A decision tree can be easily over-fitted generating too many branches and may reflect anomalies due to noise or outliers

MODULE DIAGRAM

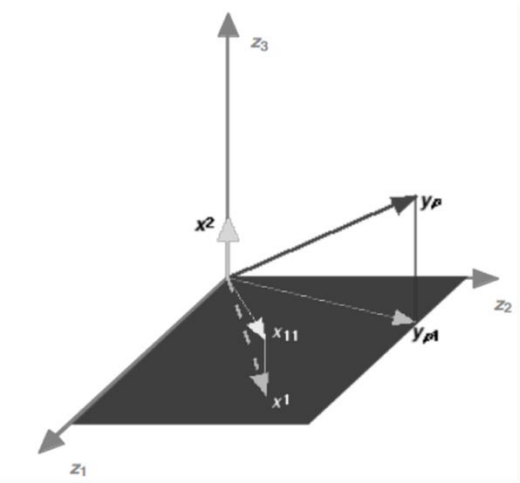


3. Gram- Schmidt orthogonalization

Gram- Schmidt orthogonalization method is used to rank features (inputs) based on the relevance of each of them versus the output (or how predictive those features are...

$$\cos^2\left(x^k, y_p\right) = \frac{\left({}^T x^k\, y_p\right)^2}{\left({}^T x^k\, x^k\right)\left({}^T y_p\, y_p\right)}$$

xk is the vector of the N measures of the kth input variable. yp is the vector of the N measurements of the output (quantity to be predicted). The Gram-Schmidt method consists in ranking the inputs by decreasing relevance level following an iterative orthogonalization method. Inputs are called descriptors. First, we choose the most correlated input with the output (by cosine calculation). Then, project the output and all others descriptors on the subspace orthogonal to the descriptor you've just selected (the most correlated with the output). Finally, to iterate the same approach from this subspace



The following function (implemented in Matlab) classifies input variables P by a Gram-Schmidt orthogonalization process to explain an output vector Z : function [index ro] = Gram_Schmidt(Z,P,nb) This function returns the vector index containing the ranking of your variables from the most to the least relevant. Z is a column vector with as many lines as examples, P is the observation matrix MxN, M columns for M variables, N lines for N examples. nb (optional) indicates the number of vectors to order.

4. "SHapley Additive exPlanations:

SHAP stands for "SHapley Additive exPlanations." Shapley values are a widely used approach from cooperative game theory. The essence of Shapley value is to measure the contributions to the final outcome from each player separately among the coalition, while preserving the sum of contributions being equal to the final outcome. When using SHAP values in model explanation, we can measure the input features' contribution to individual predictions.

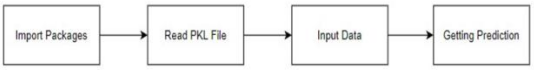
Deployment

Flask (Web Framework):

Flask is a micro web framework written in Python.It is classified as a micro-framework because it does not require particular tools or libraries.It has no database abstraction layer, form validation, or any

other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

MODULE DIAGRAM



- Flask has a lightweight and modular design, so it easy to transform it to the web framework you need with a few extensions without weighing it down
- ORM-agnostic: you can plug in your favourite ORM e.g. SQLAlchemy.
- Basic foundation API is nicely shaped and coherent.
- Flask documentation is comprehensive, full of examples and well structured. You can even try out some sample application to really get a feel of Flask.
- It is super easy to deploy Flask in production (Flask is 100% WSGI 1.0 compliant") 40
- HTTP request handling functionality • High Flexibility

The configuration is even more flexible than that of Django, giving you plenty of solution for every production need

SYSTEM SPECIFICATION

ENVIRONMENTAL REQUIREMENTS

1.Software Requirements:

Operating System : Windows

Tool : Anaconda with Jupyter Notebook

2.Hardware requirements:

Processor : Pentium IV/III

Hard disk : minimum 80 GB

RAM : minimum 2 GB

MODULES

The system module is categorized into six sub-modules namely,

Module 1: Pre-Processing

Module 2: Visualization

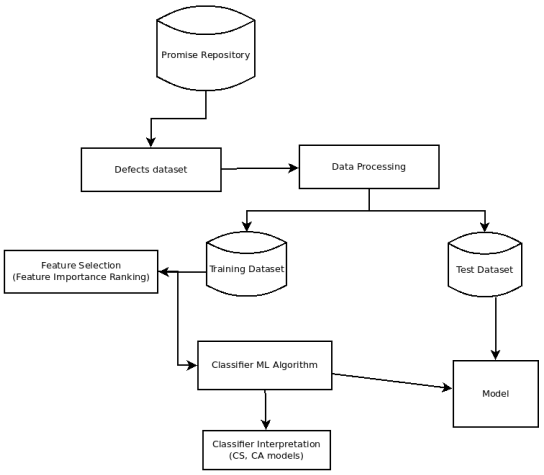
Module 3: Logistic Regression

Module 4: Random Forest Algorithm

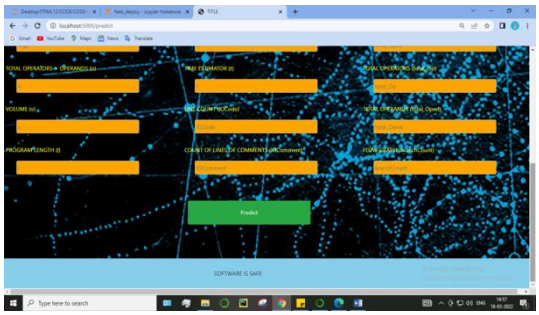
Module 5: Naive Bayes Algorithm

Module 6: K-Nearest Neighbor Algorithm

Architecture Diagram



SCREENNSHOTS



CONCLUSION

The analytical process started from data cleaning and processing, missing value, exploratory analysis and finally model building and evaluation. The best accuracy on public test set is higher accuracy score will be found out.The use of Gram-Schmidt did indeed bring about a negligible improvement on the agreeableness of CS and CA methods . It was found that there were no notable differences in between the use of sHap and Permutation algorithms for classifier interpretation although it is still subjectable to be not the same.

FUTURE ENHANCEMENT

1.Software bugs prediction to connect with AI model.

2. More diversified feature interaction removal mechanisms must be tested.

3. Further testing on the interchangeability of sHap and Permutation methods must be looked upon.

REFERENCES

[1] Romi Satria Wahono (2015), “A Systematic Literature Review of Software Defect Prediction: Research Trends, Datasets, Methods and Frameworks”, in Journal of Software Engineer Vol I

[2] Huiyang Zhou, Martin Dimitrov, (2009), “Anomaly-Based Bug Prediction, Isolation, and Validation: An Automated Approach for Software Debugging”, in ASPLOS’09 March 7-11 2009

[3] David Paterson, Gordon Fraser ,Gregory M. Kapfhammer, Jose Campos, Phil McMinn, Rui Abreu, (2019), “An Empirical Study on the Use of Defect Prediction for Test Case Prioritization”, 12th IEEE Conference on Software Testing, Validation and Verification (ICST)

[4] Yuni Rosita Dewi, Hendri Murfi, Yudi Satria, (2020) “Gram Schmidt Orthogonalization for Feature Ranking and selection – A case study of claim prediction”, in International Journal of Machine Learning and Computing.

[5] F. Zhang, A. E. Hassan, S. McIntosh, and Y. Zou, “The use of summation to aggregate software metrics hinders the performance of defect prediction models,” IEEE Transactions on Software Engineering, vol. 43, no. 5, pp. 476–491, 2016.

[6] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan, “High-impact defects: a study of breakage and surprise defects,” in Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. ACM, 2011, pp. 300–310.

[7] D. Chen, W. Fu, R. Krishna, and T. Menzies, “Applications of psychological science for actionable analytics,” in Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, 2018, pp. 456–467.

[8] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, “The impact of correlated metrics on the interpretation of defect models,” IEEE Transactions on Software Engineering, 2019.

[9] C. Theisen, K. Herzig, P. Morrison, B. Murphy, and L. Williams, “Approximating attack surfaces with stack traces,” in Proceedings of the 37th International Conference on Software Engineering-Volume 2. IEEE Press, 2015, pp. 199–208.

[10] K. Herzig, S. Just, and A. Zeller, “The impact of tangled code changes on defect prediction models,” Empirical Software Engineering, vol. 21, no. 2, pp. 303–336, 2016..

[11] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust prediction of faultproneness by random forests," in 15th international symposium on software reliability engineering. IEEE, 2004, pp. 417–428.

[12] H. Jahanshahi, D. Jothimani, A. Bas¸ar, and M. Cevik, "Does chronology matter in jit defect prediction?: A partial replication study," in Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering. ACM, 2019, pp. 90–99. 14

[13] T. Mori and N. Uchihira, "Balancing the trade-off between accuracy and interpretability in software defect prediction," Empirical Software Engineering, vol. 24, no. 2, pp. 779–825, 2019.

[14] S. Hooker, D. Erhan, P.-J. Kindermans, and B. Kim, "A benchmark for interpretability methods in deep neural networks," in Advances in Neural Information Processing Systems, 2019, pp. 9737–9748.

[15] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, "The impact of using regression models to build defect classifiers," in 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). IEEE, 2017, pp. 135–145.

[16] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," IEEE Transactions on Software Engineering, 2018.

[17] T. Menzies and M. Shepperd, "Special issue on repeatable results in software engineering prediction," 2012.

[18] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. Grundy, "An empirical study of model-agnostic techniques for defect prediction models," IEEE Transactions on Software Engineering, 2020

[19] K Z Mao, "Fast orthogonal forward selection algorithm for feature subset selection",2002.

[20] T. Yu, W. Wen, X. Han, and J. Hayes, "Conpredictor: Concurrency defect prediction in real-world applications," IEEE Transactions on Software Engineering, 2018.