HYBRID NATURE-BASED AND ENSEMBLE MACHINE LEARNING MODEL FOR SOFTWARE BUG FORECASTING

[1]*Shaik Mohammad Azaruddin, MCA Student, Department of MCA*

[2] *K Kumara Swamy, M.Tech, Assistant Professor, Department of MCA*

[12]*Dr KV Subba Reddy Institute of Technology, Dupadu, Kurnool*

https://doi.org/10.51470/ijcnwc.2025.v15.i02.pp879-888

ABSTRACT

Due to its significance in fixing the defects found in software testing, the maintenance process of software systems has drawn the attention of researchers in software development systems. BRs include information such as the bug's description, status, reporter, assignee, priority, and severity, among other details. Because the quantity of BRs grows exponentially, personally analysing them all to find system problems becomes an arduous and time-consuming ordeal, which is the primary challenge of this approach. Consequently, it is advisable to use an automated solution. The majority of the ongoing research is devoted to automating this process from various angles, such as determining the bug's importance or severity. The flaw, however, is a multi-class categorisation challenge, and they failed to account for this. This study presents a novel prediction model that can analyse BRs and accurately forecast the bug's existence, therefore resolving the issue. A combination of machine learning and natural language processing (NLP) is used to build an ensemble algorithm in the suggested model. An open-source dataset for two online software bug repositories (Eclipse and Mozilla) is used to mimic the suggested paradigm. This dataset has six categories: Program Anomaly, GUI, Network or Security, Configuration, Performance, and Test-Code. With an accuracy of 90.42% without text augmentation and 96.72% with text augmentation, the suggested model outperforms the majority of current models in the simulations.

I. INTRODUCTION

The purpose of testing in software engineering is to determine whether a system satisfies the criteria specified by the stakeholders. This assessment process includes looking for errors or failures to satisfy these objectives [1]. This procedure ensures that any issues found after testing has ended are addressed during the maintenance period. Furthermore, software developers are more likely to deploy flawed software as the product's complexity and size grow [2], and the likelihood of errors in software projects increases. As a result, consumers document the issues they've encountered [2]. When software has a defect that causes it to act improperly or provide inaccurate results, it is called a bug [3]. The reporter's comments are sent to the BTS via a bug report. The Eclipse repository contains sample problem reports, as seen in Figure 1. 1 There are many pieces of information that go into a bug report. These include the problem's ID, its status (closed or opened), a description of the issue, the program involved, details on how to recreate the bug, who reported the bug, and the developer responsible for fixing it [4].

One way to look at a bug report is as a conduit for communicating the issue to the programmers working on the solution [5]. After receiving a problem report, the developer follows a certain procedure known as the bug management process [6] to fix the issue. When consumers encounter an issue with a published software product and file a complaint to the bug management system, this procedure begins. Developers are then tasked with investigating this problem report. When a developer discovers the source and location of a defect before other developers do, they are the ones who repair it. Following the bug's resolution, the tester verifies that the problem has not resurfaced by checking the bug scenario. If it hasn't, the status of the bug report is updated to Verified. At long last, the reporter gets a notice [6].

There are several stages that make up a software bug's life cycle. Hence, the bug report life cycle is shown in Figure 2. According to this diagram, there are three stages to a bug's life cycle: management, triage, and localisation. From the moment a user reports an issue until the developer is assigned, everything that happens during this period is known as bug management. next the prioritisation and assignment of appropriate developers to fix submitted reports, the next process is bug triage. Phase three, bug localisation, is responsible for transitioning the bug status from resolved to verified and finally closed [7]. The exponential growth in the volume of reported bugs is the biggest obstacle to overcome throughout this life cycle since handling them manually is a painstaking, complicated, and time-consuming process [8]. To solve this problem, researchers sort the reports into three primary groups, each with its own set of subgroups [9], and then extract relevant data to help speed up and simplify the maintenance step. These types of problem reports are sorted according to priority, severity, and kind [9]. The majority of research sorts reported bugs according to priority or severity.

This research's literature study reveals that different classification algorithms categorise bug reports based on different criteria; nonetheless, there is a dearth of highly accurate bug classification models that are based on nature. So, to fill this need, this research introduces an ensemble machine learning approach for bug reporting that is based on nature for bug prediction.

The goal of this study is to automate the process of predicting the sorts of bugs in software systems via the use of natural language processing (NLP), machine learning (ML), and text mining methods. Instead of spending time manually locating bugs, the model may accomplish it quickly during the maintenance phase by using bug localisation methods. Figure 3 shows a high-level perspective of how a bug report is used to anticipate future issues.

II. LITERATURE SURVEY

"Software testing techniques: A literature review," by M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad,

Quality assurance of created software has reached new heights due to the rising complexity of modern software applications and the intense competition in the market. Given the importance of software testing both before and after development, it is imperative that the Software Development Lifecycle include more effective and efficient procedures and approaches to ensure quality software testing. The goal of this article is to talk about current and new testing methods for better quality assurance.

The "Deep neural network-based severity prediction of bug reports" was written by W. Y. Ramay, Q. Umer, X. C. Yin, C. Zhu, and I. Illahi.

The maintenance phase is a crucial part of developing software. In order to fix software issues, developers use issue tracking systems. Such issue tracking systems allow users to report issues and assign severity ratings to them. The urgency with which an issue should be fixed is determined by its severity. Timely resolution of critical problems is facilitated for developers. Manual severity evaluation, on the other hand, is time-consuming and prone to error. In this research, we provide an automated method for predicting the severity of bug reports that is based on deep neural networks. After collecting bug reports, the first step is to preprocess the text using natural language processing algorithms. Our second step is to give each bug report a score based on how they felt about it. Next, we construct a vector for every bug report that has been preprocessed. Finally, a classifier based on a deep neural network is used to estimate the severity of each bug report. This classifier is fed the created vector together with the emotion score of each report. Additionally, we assess the suggested method using bug report history data. The cross-product findings seem like the suggested method beats the state-of-the-art methods. It enhances the f-measure by an average of 7.90%.

"A technique of non-bug report identification from bug report repository," by J. Polpinij (2013) Artif. Robot for Life

Studies on bug reports often deal with the problem of misclassification, which occurs when non-bug reports are filtered out of the repository after being identified. The time and effort spent triaging and fixing issues increases as a result of having to filter out irrelevant complaints, which in turn loses time in discovering real bug reports. Consequently, this matter has been thoroughly investigated and is discussed below. In order to address this issue, this research suggests a way to automatically detect reports that do not include bugs in the bug report repository by using classification algorithms. Here, three things are taken into account. To begin, unigram and CamelCase are used as bug report features, with CamelCase words being utilised for feature expansion. We then analyse five different word weighting strategies to find the one that works best for this job. One class support vector machine (SVM) based on Schölkopf technique, binary-class support vector machine (SVM), and support vector data description (SVDD) are the major methods used for modelling non-bug report identifiers. Following recall, precision, and F1 testing, the results show that the bug report repository can efficiently identify reports that do not include bugs. After comparing our results to those of other prominent research, our findings may be considered acceptable. Specifically, we found that the Scölkopf approach and SVDD methods performed the best when using non-bug report IDs with tf-igm and modified tf-icf weighting schemes.

S. Adhikarla, "Bug report routing using automated bug classification," Juris Doctor thesis, Faculty of Arts and Sciences, Department of Computer Science, Linköping University, Linköping, Sweden, 2020.

As software technology advances, businesses increasingly look to automated solutions as a means to save costs and increase efficiency. The software business has seen enormous development in automated solutions, thanks to the substantial machine learning research that has supported them. While there has been a lot of study into automated bug categorisation, there is always a need for more accurate approaches that take into account the newly acquired data. After reading the defect report, an automatic bug classifier will determine which department or individual is responsible for fixing the issue.

Typically, a bug report will include an unstructured text box where the issue may be explained in depth. The topic of data extraction from these types of text fields has been extensively studied. In order to extract two characteristics from the bug report's unstructured text fields, this thesis use a topic modelling approach called Latent Dirichlet Allocation (LDA) and a numerical statistic called Term Frequency - Inverse Document Frequency (TF-IDF). By merging the TF-IDF and LDA features, a third set of features was generated. Over the course of this thesis, the data utilised undergoes a transformation in its class distribution. As a feature, the age of the bug report was incorporated to examine the influence of time on the prediction. Along with the LDA and TF-IDF characteristics, this feature's significance was investigated in this research.

Three distinct classification models—DO-probit, dense neural networks, and multinomial logistic regression—were trained using these produced feature vectors as predictors. The accuracy and F1-score of the classifiers' predictions for which department should address an issue were assessed. Predictions made by a Support Vector Machine (SVM) with a linear kernel served as the reference point for this comparison.

III. SYSTEM ANALYSIS AND DESIGN

EXISTING SYSTEM

Kukkar and Mohana [10] address the issue of misclassification of bug reports—which impacts the overall effectiveness of the prediction process—by using text mining, natural language processing, and machine learning approaches to categorise bug and non-bug reports. Bigram and TF-IDF are used for feature selection in this model. But as the dataset changes, the KNN algorithm's performance in this model also changes.

When it comes to bug reporting, some researchers have various criteria. In [11], researchers use support vector machines (SVMs), naïve bayes (NBs), and random trees (RTs) as machine learning techniques to categorise newly reported bugs as either perfective (requiring significant upkeep) or corrective (requiring defect correction). With a score of 93.1%, SVM proved to be the most accurate method.

A popular model known as Orthogonal Defect Classification (ODC) was established by researchers to aid in the software engineering process [12]. Several analytical approaches for test process analysis and software development are available in ODC, which contains eight orthogonal properties to characterise software problems [51]. Using ODC, we may extract useful information from faults, which aids in software engineering process diagnostics, and gives us new insights [13]. To categorise bug reports in accordance with ODC, the writers in [13] use machine learning methods. A total of 4096 bug reports tagged with ODC were used in their investigation. They do, however, come to the conclusion that bug reports alone are insufficient for automating the ODC properties.

Hirsch and Hofer [14] used three categories—concurrency, memory, and semantic bugs—to categorise each newly reported problem. Using 369 reports of bugs, they achieved a maximum mean recall of 0.72 and a precision of 0.74.

DISADVANTAGES

- An existing system is not hybrid deep learning detection policy to improve the efficiency and effectiveness of Bug Report Generation.
- An existing system never used Nature-Based Ensemble Machine Learning Bug Prediction Model which is more accurate and efficient.

PROPOSED SYSTEM

• It offers a review of many foundational ML techniques for automated categorisation of bug reports based on nature.

• It suggests a method for automatically predicting bugs based on nature using bug reports and an ensemble machine learning methodology.

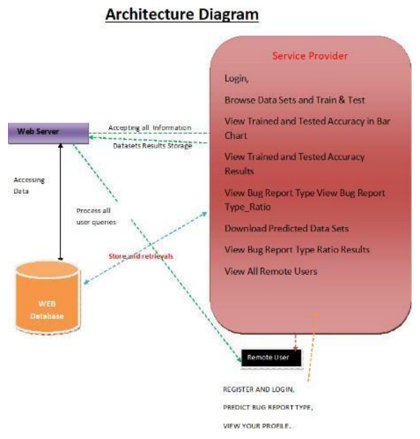
• The suggested approach incorporates a text augmentation strategy for nature-based issue prediction using bug reports. As far as we are aware, this is the first ensemble machine learning method to use a text augmentation methodology to forecast the kind of defects in bug reports.

• The findings of the proposed technique's assessment on a benchmark dataset indicate that the ensemble machine learning approach is effective in predicting bugs based on nature from reports of bugs.

Advantages

- The proposed algorithm aims to enhance nature-based bug prediction by using several machine learning (ML) base classifiers and training them using a benchmark dataset.
- The proposed algorithm integrates machine learning, NLP, and text-mining techniques. Their algorithm uses 504 bug reports and classifies them into four classes: assignment/initialization, external interface, internal interface, and other. This model achieved 73.70% of accuracy.

SYSTEM ARCHITECTURE


Architecture Diagram
Service Provider
Login,
Browse Data Sets and Train & Test
View Trained and Tested Accuracy in Bar Chart
View Trained and Tested Accuracy Results
View Bug Report Type View Bug Report Type_Ratio
Download Predicted Data Sets
View Bug Report Type Ratio Results
View All Remote Users
Web Server
Accepting all Information
Datasets Results Storage
Accessing Data
Process all user queries
Store and retrievals
WEB Database
Remote User
REGISTER AND LOGIN,
PREDICT BUG REPORT TYPE,
VIEW YOUR PROFILE.

IV. IMPLEMENTATION

Modules

Service Provider

A valid username and password are required for the Service Provider to access this module. Following a successful login, he will have access to many activities, including the ability to browse data sets and perform train and test. Check out the Bar Chart for Trained and Tested Accuracy, Check Out the Results for Trained and Tested Accuracy, See the Type of Bug Report, the Ratio of Bug Reports, Download Data Sets for Prediction, and See the Results of the Ratio of Bug Reports, See Who Is Online From Afar.

View and Authorize Users

The admin can get a complete rundown of all registered users in this section. Here, the administrator may see the user's information (name, email, and address) and grant them access.

Remote User

A total of n users are present in this module. Before beginning any actions, the user needs register. Following registration, the user's information will be entered into the database. Following a successful registration, he must use his password and authorised user name to log in. Following a successful login, the user will be able to see their profile, predict the kind of bug report, and register and log in.

ALGORITHM

Logistic regression Classifiers

The relationship between a collection of independent (explanatory) factors and a categorical dependent variable is examined using logistic regression analysis. When the dependent variable simply has two values, like 0 and 1 or Yes and No, the term logistic regression is used. When the dependent variable contains three or more distinct values, such as married, single, divorced, or widowed, the technique is sometimes referred to as multinomial logistic regression. While the dependent variable's data type differs from multiple regression's, the procedure's practical application is comparable. When it comes to categorical-response variable analysis, logistic regression and discriminant analysis are competitors. Compared to discriminant analysis, many statisticians believe that logistic regression is more flexible and appropriate for modelling the majority of scenarios. This is due to the fact that, unlike discriminant analysis, logistic regression does not presume that the independent variables are regularly distributed.

Both binary and multinomial logistic regression are calculated by this software for both category and numerical independent variables. Along with the regression equation, it provides information on likelihood, deviance, odds ratios, confidence limits, and quality of fit. It does a thorough residual analysis that includes diagnostic residual plots and reports. In order to find the optimal regression model with the fewest independent variables, it might conduct an independent variable subset selection search. It offers ROC curves and confidence intervals on expected values to assist in identifying the optimal classification cutoff point. By automatically identifying rows that are not utilised throughout the study, it enables you to confirm your findings.

Naïve Bayes

The supervised learning technique known as the "naive bayes approach" is predicated on the straightforward premise that the existence or lack of a certain class characteristic has no bearing on the existence or nonexistence of any other feature.

However, it seems sturdy and effective in spite of this. It performs similarly to other methods of guided learning. Numerous explanations have been put forward in the literature. We emphasise a representation bias-based explanation in this lesson. Along with logistic regression, linear discriminant analysis, and linear SVM (support vector machine), the naive bayes classifier is a linear classifier. The technique used to estimate the classifier's parameters (the learning bias) makes a difference.

Although the Naive Bayes classifier is commonly used in research, practitioners who want to get findings that are useful do not utilise it as often. On the one hand, the researchers discovered that it is very simple to build and apply, that estimating its parameters is simple, that learning occurs quickly even on extremely big datasets, and that, when compared to other methods, its accuracy is rather excellent. The end users, however, do not comprehend the value of such a strategy and do not get a model that is simple to read and implement.

As a consequence, we display the learning process's outcomes in a fresh way. Both the deployment and comprehension of the classifier are simplified. We discuss several theoretical facets of the naive bayes classifier in the first section of this lesson. Next, we use Tanagra to apply the method on a dataset. We contrast the outcomes (the model's parameters) with those from other linear techniques including logistic regression, linear discriminant analysis, and linear support vector machines. We see that the outcomes are quite reliable. This helps to explain why the strategy performs well when compared to others. We employ a variety of tools (Weka 3.6.0, R 2.9.2, Knime 2.1.1, Orange 2.0b, and RapidMiner 4.6.0) on the same dataset in the second section. Above all, we make an effort to comprehend the outcomes.

Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for decision trees' habit of overfitting to their training set. Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance.

The first algorithm for random decision forests was created in 1995 by Tin Kam Ho[1] using the random subspace method, which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.

An extension of the algorithm was developed by Leo Breiman and Adele Cutler, who registered "Random Forests" as a trademark in 2006 (as of 2019, owned by Minitab, Inc.).The extension combines Breiman's "bagging" idea and random selection of features, introduced first by Ho[1] and later independently by Amit and Geman[13] in order to construct a collection of decision trees with controlled variance.

Random forests are frequently used as "blackbox" models in businesses, as they generate reasonable predictions across a wide range of data while requiring little configuration.

SVM

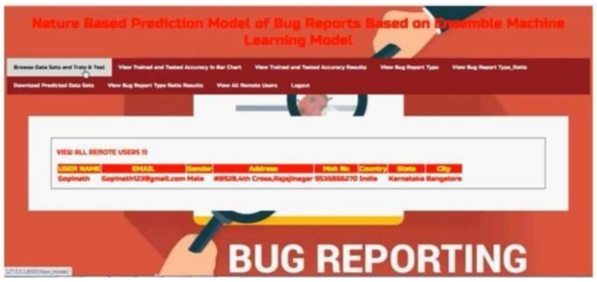
In classification tasks a discriminant machine learning technique aims at finding, based on an independent and identically distributed (iid) training dataset, a discriminant function that can correctly predict labels for newly acquired instances. Unlike generative machine learning approaches, which require computations of conditional probability distributions, a discriminant classification function takes a data point x and assigns it to one of the different classes that are a part of the classification task. Less powerful than generative approaches, which are mostly used when prediction involves outlier detection, discriminant approaches require fewer computational resources and less training data, especially for a multidimensional feature space and when only posterior probabilities are needed. From a geometric perspective, learning a classifier is equivalent to finding the equation for a multidimensional surface that best separates the different classes in the feature space.

SVM is a discriminant technique, and, because it solves the convex optimization problem analytically, it always returns the same optimal hyperplane parameter—in contrast to genetic algorithms (GAs) or perceptrons, both of which are widely used for classification in machine learning. For perceptrons, solutions are highly dependent on the initialization and termination criteria. For a specific kernel that transforms the data from the input space to the feature space, training returns uniquely defined SVM model parameters for a given training set, whereas the perceptron and GA classifier models are different each time training is initialized. The aim of GAs and perceptrons is only to minimize error during training, which will translate into several hyperplanes' meeting this requirement.

V. SCREEN SHOTS

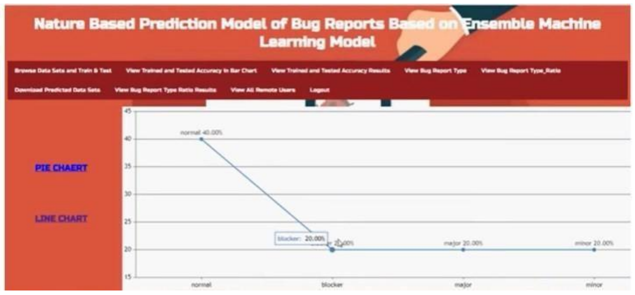
Nature Based Prediction Model of Bug Reports Based on Ensemble Machine Learning Model
Login Now
User Name
Password
sign_in
SERVICE PROVIDER
REGISTER
BUG REPORTING

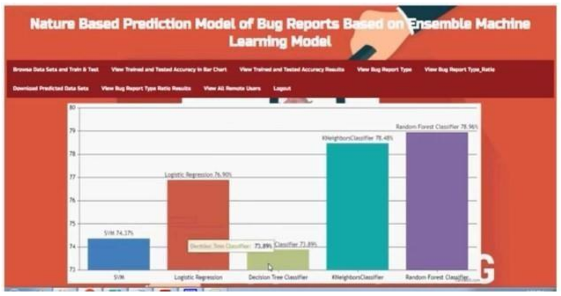
Nature Based Prediction Model of Bug Reports Based on Ensemble Machine Learning Model
Login Now
Login Service Provider
Admin
Password
Login
User Login
BUG REPORTING

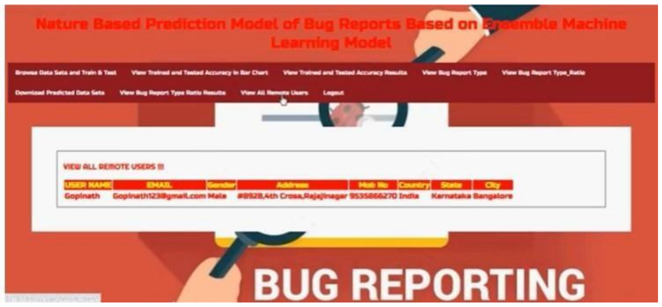

Nature Based Prediction Model of Bug Reports Based on Ensemble Machine Learning Model
Browse Data Sets and Train & Test
View Trained and Tested Accuracy in Bar Chart
View Trained and Tested Accuracy Results
View Bug Report Type
View Bug Report Type_Ratio
Download Predicted Data Sets
View Bug Report Type Ratio Results
View All Remote Users
Logout
BUG REPORTING

VIEW ALL REMOTE USERS !!!

USER NAME	EMAIL	Gender	Address	Mob No	Country	State	City
Gopinath	Gopinath123@gmail.com	Male	#8928,4th Cross,Rajajinagar	9535866270	India	Karnataka	Bangalore

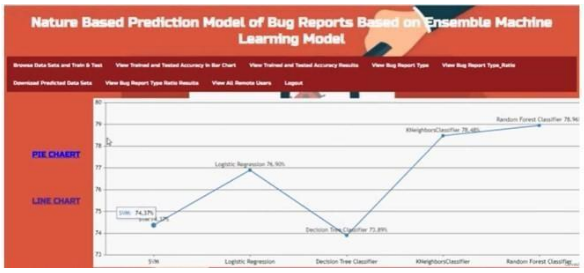


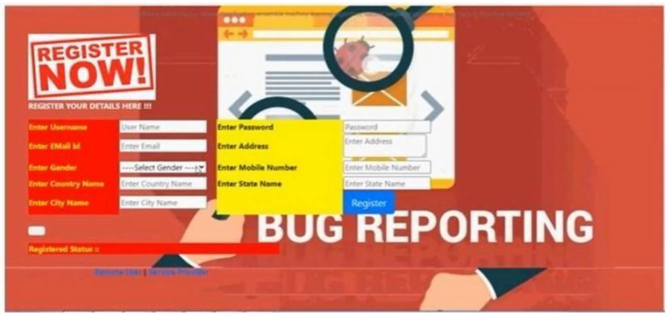



Nature Based Prediction Model of Bug Reports Based on Ensemble Machine Learning Model
BUG REPORTING

VIEW ALL REMOTE USERS !!!

USER NAME	EMAIL	Gender	Address	Mob No	Country	State	City
Gopinath	Gopinath123@gmail.com	Male	#8928,4th Cross,Rajajinagar	9535866270	India	Karnataka	Bangalore

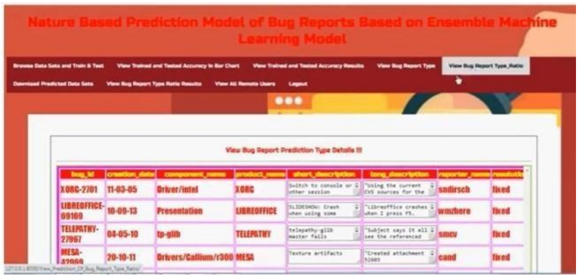




REGISTER YOUR DETAILS HERE !!!

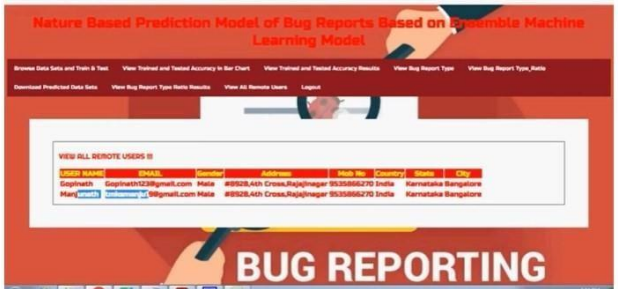
Enter Username	User Name	Enter Password	Password
Enter EMail Id	Enter Email	Enter Address	Enter Address
Enter Gender	----Select Gender ----	Enter Mobile Number	Enter Mobile Number
Enter Country Name	Enter Country Name	Enter State Name	Enter State Name
Enter City Name	Enter City Name		

Registered Status ::


Nature Based Prediction Model of Bug Reports Based on Ensemble Machine Learning Model

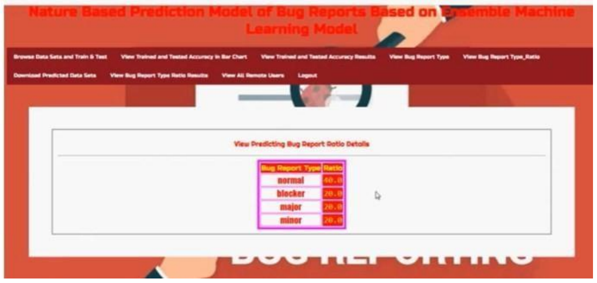
View Bug Report Prediction Type Details !!!

bug_id	creation_date	component_name	product_name	short_description	long_description	reporter_name	resolutio
XORG-2701	11-03-05	Driver/intel	XORG	Switch to console or other session	"Using the current CVS sources for the	sndirsch	fixed
LIBREOFFICE-69169	10-09-13	Presentation	LIBREOFFICE	SLIDESHOW: Crash when using some	"Libreoffice crashes when I press F5.	wmzhere	fixed
TELEPATHY-27967	04-05-10	tp-glib	TELEPATHY	telepathy-glib master fails	"Subject says it all see the referenced	smcv	fixed
MESA-42069	20-10-11	Drivers/Gallium/r300	MESA	Texture artifacts	"Created attachment 52603	cand	fixed


Nature Based Prediction Model of Bug Reports Based on Ensemble Machine Learning Model
BUG REPORTING

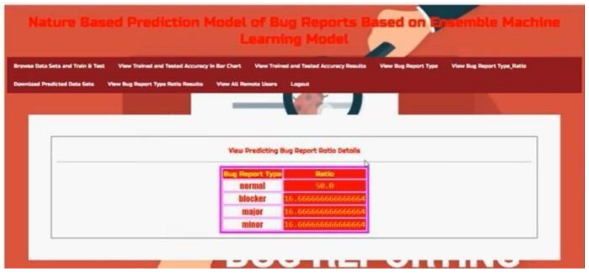
VIEW ALL REMOTE USERS !!!

USER NAME	EMAIL	Gender	Address	Mob No	Country	State	City
Gopinath	Gopinath123@gmail.com	Male	#8928,4th Cross,Rajajinagar	9535866270	India	Karnataka	Bangalore
Manjunath	[illegible]98@gmail.com	Male	#8928,4th Cross,Rajajinagar	9535866270	India	Karnataka	Bangalore


Nature Based Prediction Model of Bug Reports Based on Ensemble Machine Learning Model

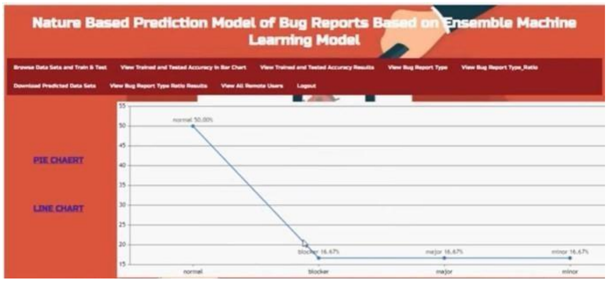
View Predicting Bug Report Ratio Details

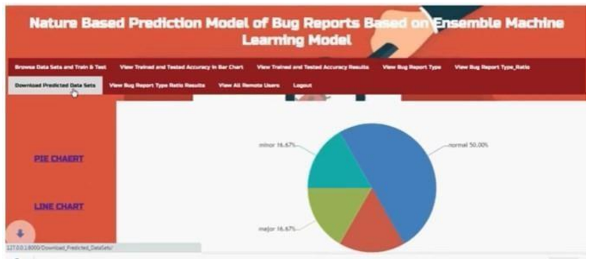
Bug Report Type	Ratio
normal	60.0
blocker	20.0
major	20.0
minor	20.0


Nature Based Prediction Model of Bug Reports Based on Ensemble Machine Learning Model

View Predicting Bug Report Ratio Details

Bug Report Type	Ratio
normal	50.0
blocker	16.666666666666664
major	16.666666666666664
minor	16.666666666666664


Nature Based Prediction Model of Bug Reports Based on Ensemble Machine Learning Model
Browse Data Sets and Train & Test
View Trained and Tested Accuracy in Bar Chart
View Trained and Tested Accuracy Results
View Bug Report Type
View Bug Report Type_Ratio
Download Predicted Data Sets
View Bug Report Type Ratio Results
View All Remote Users
Logout
PIE CHAERT
LINE CHART
55
50
45
40
35
30
25
20
15
normal 50.00%
blocker 16.67%
major 16.67%
minor 16.67%
normal
blocker
major
minor


Nature Based Prediction Model of Bug Reports Based on Ensemble Machine Learning Model
Browse Data Sets and Train & Test
View Trained and Tested Accuracy in Bar Chart
View Trained and Tested Accuracy Results
View Bug Report Type
View Bug Report Type_Ratio
Download Predicted Data Sets
View Bug Report Type Ratio Results
View All Remote Users
Logout
PIE CHAERT
LINE CHART
minor 16.67%
normal 50.00%
major 16.67%
127.0.0.1:8000/Download_Predicted_DataSets/

VI. CONCLUSION

An ensemble machine learning approach including four foundational machine learning algorithms—Random Forest, Support Vector Classification, Logistic Regression, and Multinomial Naïve Bayes—was used in this study to propose a nature-based bug prediction component. The model's accuracy is 90.42%. To improve accuracy, it also makes use of a text augmentation approach. As a result, the suggested model's maximum accuracy rose to 96.72%. Six problem categories—Program Anomaly, GUI, Network or Security, Configuration, Performance, and Test-Code—are used by the suggested model to anticipate the kind of issue. This model will be improved in future work by adding more problem categories and suggesting potential fixes for anticipated issues to save down on maintenance effort.

REFERENCES

[1] M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, ‘‘Software testing techniques: A literature review,’’ in Proc. 6th Int. Conf. Inf. Commun. Technol. Muslim World (ICT4M), Nov. 2016, pp. 177–182.

[2] W. Y. Ramay, Q. Umer, X. C. Yin, C. Zhu, and I. Illahi, ‘‘Deep neural network-based severity prediction of bug reports,’’ IEEE Access, vol. 7, pp. 46846–46857, 2019.

[3] W. Wen, ‘‘Using natural language processing and machine learning techniques to characterize configuration bug reports: A study,’’ M.S. thesis, College Eng., Univ. Kentucky, Lexington, KY, USA, 2017.

[4] J. Polpinij, ‘‘A method of non-bug report identification from bug reportrepository,’’ Artif. Life Robot., vol. 26, no. 3, pp. 318–328, Aug. 2021.

[5] S. Adhikarla, ‘‘Automated bug classification.: Bug report routing,’’M.S. thesis, Fac. Arts Sci., Dept. Comput. Inf. Sci., Linköping Univ.,Linköping, Sweden, 2020.

[6] K. C. Youm, J. Ahn, and E. Lee, ‘‘Improved bug localization based oncode change histories and bug reports,’’ Inf. Softw. Technol., vol. 82, pp. 177–192, Feb. 2017.

[7] N. Safdari, H. Alrubaye, W. Aljedaani, B. B. Baez, A. DiStasi, and M. W. Mkaouer, ‘‘Learning to rank faulty source files for dependent bugreports,’’ in Proc. SPIE, vol. 10989, 2019, Art. no. 109890B.

[8] A. Kukkar, R. Mohana, A. Nayyar, J. Kim, B.-G. Kang, and N. Chilamkurti, ‘‘A novel deep-learning-based bug severity classification technique using convolutional neural networks and random forest with boosting,’’ Sensors, vol. 19, no. 13, p. 2964, Jul. 2019.

[9] A. Aggarwal. (May 2020). Types of Bugs in Software Testing: 3 Classifications With Examples. [Online]. Available: https://www.scnsoft.com/software-testing/types-of-bugs

[10] A. Kukkar and R. Mohana, ‘‘A supervised bug report classification with incorporate and textual field knowledge,’’ Proc. Comput. Sci., vol. 132, pp. 352–361, Jan. 2018.

[11] A. F. Otoom, S. Al-jdaeh, and M. Hammad, ‘‘Automated classification of software bug reports,’’ in Proc. 9th Int. Conf. Inf. Commun. Manage., Aug. 2019, pp. 17–21.

[12] P. J. Morrison, R. Pandita, X. Xiao, R. Chillarege, and L. Williams, ‘‘Are vulnerabilities discovered and resolved like

other defects?'' Empirical Softw. Eng., vol. 23, no. 3, pp. 1383–1421, Jun. 2018.

[13] F. Lopes, J. Agnelo, C. A. Teixeira, N. Laranjeiro, and J. Bernardino, ''Automating orthogonal defect classification using machine learning algorithms,'' Future Gener. Comput. Syst., vol. 102, pp. 932–947, Jan. 2020.

[14] T. Hirsch and B. Hofer, ''Root cause prediction based on bug reports,'' in Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops (ISSREW), Oct. 2020, pp. 171–176.

[15] Q. Umer, H. Liu, and I. Illahi, ''CNN-based automatic prioritization of bug reports,'' IEEE Trans. Rel., vol. 69, no. 4, pp. 1341–1354, Dec. 2020.

[16] H. Bani-Salameh, M. Sallam, and B. Al Shboul, ''A deep-learning-based bug priority prediction using RNN-LSTM neural,'' e-Inform. Softw. Eng. J., vol. 15, no. 1, pp. 1–17, 2021.

[17] Ö. Köksal and B. Tekinerdogan, ''Automated classification of unstructured bilingual software bug reports: An industrial case study research,'' Appl. Sci., vol. 12, no. 1, p. 338, Dec. 2021.

[18] B. Alkhazi, A. DiStasi, W. Aljedaani, H. Alrubaye, X. Ye, and M. W. Mkaouer, ''Learning to rank developers for bug report assignment,'' Appl. Soft Comput., vol. 95, Oct. 2020, Art. no. 106667.

[19] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, ''Automated bug assignment: Ensemble-based machine learning in largescale industrial contexts,'' Empirical Softw. Eng., vol. 21, no. 4, pp. 1533–1578, Aug. 2016.

[20] X. Ye, R. Bunescu, and C. Liu, ''Learning to rank relevant files for bug reports using domain knowledge,'' in Proc. 22nd ACMSIGSOFT Int. Symp. Found. Softw. Eng., Nov. 2014, pp. 689–699.

[21] Y. Tian, D. Wijedasa, D. Lo, and C. Le Goues, ''Learning to rank for bug report assignee recommendation,'' in Proc. IEEE 24th Int. Conf. Program Comprehension (ICPC), May 2016, pp. 1–10.

[22] D. Devaiya, Castr: AWeb-Based Tool for Creating Bug Report Assignment Recommenders. Lethbridge, AB, Canada: Univ. Lethbridge, 2019.

[23] M. Alenezi, S. Banitaan, and M. Zarour, ''Using categorical features in mining bug tracking systems to assign bug reports,'' 2018, arXiv:1804.07803.

[24] H. A. Ahmed, N. Z. Bawany, and J. A. Shamsi, ''CaPBug-a framework for automatic bug categorization and prioritization using NLP and machine learning algorithms,'' IEEE Access, vol. 9, pp. 50496–50512, 2021.

[25] R.-M. Karampatsis and C. Sutton, ''How often do single-statement bugs occur?: The ManySStuBs4J dataset,'' in Proc. 17th Int. Conf. Mining Softw. Repositories, Jun. 2020, pp. 573–577, doi: 10.1145/3379597.3387491.