Bug Watch: A Comprehensive Approach to Classification and Redundancy Detection in Software Testing

Dr. Sandip Shinde
Vishwakarma Institute of Technology
Pune, India
sandip.shinde@vit.edu

Sachin Chat
Dassault Medidata
California, United States
sachinchat@gmail.com

Pavan.R.Maske
Vishwakarma Institute Of Technology
Pune, India
pavan.maske21@vit.edu

Mahi Chrungoo
Vishwakarma Institute of Technology
Pune, India
mahi.chrungoo21@vit.edu

Aditya.N.Mane
Vishwakarma Institute of Technology
Pune, India
aditya.mane21@vit.edu

Sakshi Kulkarni
Vishwakarma Institute of Technology
Pune, India
sakshi.kulkarni21@vit.edu

Abstract: Countless software programs are created daily, leading to a parallel increase in software bugs. Each institution typically maintains a dedicated quality assurance team tasked with ensuring the integrity of their developed products. However, the sheer volume of bug reports received on a daily basis presents a formidable challenge for these teams to manually address each issue.

The primary aim of this research is to streamline the process of bug classification and mitigate redundancy within reported bug data. To achieve this goal, we explored the utilization of different classification algorithms, specifically SGD and Random Forest, based on the issue summary of each bug. Our approach involves categorizing bugs into two main groups: major and minor. Additionally, a redundancy removal process was implemented to preprocess the data, flagging sentences that exhibit significant similarity above a predetermined threshold as duplicates.

The results of our study demonstrate promising outcomes, with classification accuracies of 75.93% (Feature Classification) and 90.02% (Redundancy detection) achieved using SGD classifier, which gave us the best accuracy.

Keywords- Bug Report, Classification, Machine Learning, SGD classifier, Software Testing, Natural Language Processing, Duplicate detection

I. Introduction

Each day witnesses the birth and evolution of numerous software systems, a process integral to the modern digital landscape. Yet, within the dynamic realm of Software Development Life Cycle (SDLC), a notable influx of bugs surfaces during operational phases. A bug report serves as a crucial document encapsulating these issues, often accompanied by error logs or summarized problem descriptions. Upon receiving such reports, software teams embark on the task of bug resolution, employing methods ranging from code reviews to manual debugging. Among these reported bugs, some emerge as critical, necessitating immediate attention. To effectively manage these issues, it becomes imperative to categorize bugs based on the software's features, enabling focused efforts on addressing major software vulnerabilities. Moreover, the recurrence of identical issues reported by multiple users or testing teams introduces redundancy in data, amplifying the workload and impeding efficiency. This paper proposes a dual-pronged approach to address these challenges, encompassing bug classification and redundancy detection methodologies.

The bug classification framework comprises two distinct models: "Category 1," identifying the specific feature implicated in the bug, and "Category 2," characterizing the type of bug (e.g., Functional, UI). Leveraging machine learning algorithms such as SGD and Random Forest, this classification endeavor aims to enhance bug resolution efficacy.Concurrently, the redundancy removal utility offers a systematic means to identify and eliminate duplicate entries within bug reports. This process involves meticulous data preprocessing, including stopwords identification and removal, followed by similarity assessment. Instances surpassing a predefined similarity threshold are flagged as redundant data, streamlining bug report management.

For training and testing these models, a dataset encompassing software-related information was utilized, supplemented by

generated data to bolster analytical robustness. Through these methodologies, this research endeavors to fortify bug resolution frameworks, fostering enhanced software reliability and efficiency.

II. Literature Review

To help us grasp various approaches and techniques, we conducted a thorough research assessment of 14 such works. This section contains reviews of all the papers. In the paper [1], Rayhanur Rahman et al. focused on addressing specific types of bugs, namely security and performance issues. Their research introduces a learning-based methodology aimed at identifying bugs belonging to these categories. They propose two distinct classification models: the Sec-Model and the Perf-Model.The Sec-Model is designed to classify bugs as either security-related or not, while the Perf-Model focuses on identifying performance-related bugs. These models utilize machine learning techniques to analyze bug reports and make accurate classifications. To evaluate the performance of these models, the researchers utilized the Area Under Curve (AUC) metric, a common measure of classification accuracy. The AUC values obtained for the Sec-Model and Perf-Model were found to be 0.67 and 0.71, respectively. These AUC values indicate the effectiveness of the models in accurately classifying bugs based on their security and performance characteristics. John Wu et al. in the paper [2] devised a bug report classification model using an LSTM network to categorize reports as helpful or unhelpful. Their system aims to reduce false positives and enhance ranking performance, ultimately improving information retrieval models. This model strikes a balance between precision and recall, crucial for effective bug report management. In the paper, [3], Hassan Thair et al. introduced an automated bug prioritization and assignment technique named LCBPA (Long Short Term Memory, Content-Based Filtering for Bug Prioritization). Their approach combines LSTM (Long Short Term Memory) for bug prioritization and content-based filtering for bug assignment. Through experimentation, the researchers demonstrated that the LCBPA technique surpasses existing bug prioritization methods and effectively addresses the bug assignment challenge when compared to current techniques. This highlights the effectiveness and superiority of their proposed approach in bug management and resolution processes. The paper, [4], presents an automated bug report detection and classification model leveraging deep learning methodologies. The system is structured into three key modules: Preprocessing, Deep Learning Model, and Duplicate Bug Report Detection and Classification. In the Preprocessing module, textual data undergoes initial processing steps to prepare it for analysis. Following this, the Deep Learning Model employs Convolutional Neural Networks (CNNs) to extract pertinent features from the bug reports. These features are then utilized to identify similarities among bug reports.The evaluation of the system revealed an impressive accuracy range, ranging from 85% to 99%. This underscores the effectiveness of the proposed model in accurately detecting and classifying bug reports, showcasing the potential of deep learning techniques in bug management and resolution processes. [5] Salahuddin Shaikh et al. conducted a study focusing on a software defect-prone model reliant on classification techniques, specifically LibSVM and LibLinear. The research highlights that LibSVM significantly enhances both accuracy and efficiency, particularly in training set scenarios. Notably, the True Positive Rate (TP-Rate) and F-Measure experience substantial improvements compared to alternative techniques. The study concludes that LibSVM stands out for its rapid training dataset utilization, offering notable advantages in terms of accuracy and efficiency. Additionally, both LibSVM and LibLinear demonstrate prowess in enhancing various evaluation measures, further contributing to their effectiveness in software defect prediction models. [6] This paper assesses bi-grams and feature selection for bug severity prediction. Results show bi-grams can improve or worsen accuracy, which is dataset-specific. Feature selection enhances accuracy and provides valuable terms for experts, though interpretation can be difficult. Future work includes testing additional classifiers and considering temporal factors in cross-validation. [7] The approach discussed in the paper predicts bug report root causes to aid in selecting debugging tools and creating benchmarks for IR-based fault localization. Future work includes linking fault patterns to code smells, improving stack trace preprocessing, refining predictions to sub-categories, and using SMOTE to balance the training set. [8] This paper introduces a time-efficient double-tier duplicate bug report detection system using LDA-based topic modeling and classification. It outperforms conventional methods in processing time and Top-N recall rate. Future work aims to enhance recall rates, document similarity measures, and representation techniques. [9] This paper addresses the problem of determining bug severity levels through machine learning techniques, focusing on the textual summary of bug reports. The authors applied various machine learning methods and evaluated performance using precision, recall, F-Measure, and accuracy through 5-fold cross-validation. The study found that using 125 terms, selected via information gain criteria, stabilizes performance. The best F-Measures were achieved for severity levels 2, 3, and 4 across most techniques. Simultaneously, this paper, [10] presents a Bayesian network combined with the Bert model for classifying social governance texts. The Bayesian network's accuracy is lower, but the Bert model has a disadvantage. To improve text classification accuracy, the Bayesian network uses word-category reasoning, extracting the text of "ᵡ□è簡学" first, and then classifying it using Bert. The experimental results confirm the Bayesian network's effectiveness and feasibility in text classification. [11] The paper presents program slicing metrics, a set of code metrics for C programs that measure the size, complexity, coupling, and cohesion properties of programs based on program slices, vertices, and dependency edges. The study evaluates PS metrics and Understand for C++ metrics for classifying buggy/bug-free source code files or functions for 887 revisions of the Apache HTTP project and 76 revisions of the Latex2rtf project. Results show PS metrics have slightly better performance than UC metrics in bug classification. [12] This study presents a method for detecting duplicate bug reports using contextual information

from software-engineering textbooks and project documentation. The method takes half an hour to perform, with slightly lower accuracy and kappa scores than labeled LDA. It performs at par with unsupervised LDA on Eclipse, Mozilla, and Open Office, and requires no sophisticated tools or parameter optimization. The method can be applied across various software projects and requires constant updating of the LDA context. The method's robustness is demonstrated by its ability to find applications in other software-engineering tasks like feature location or concept location. [13] The paper proposes a DURFEX approach for bug tracking systems, which automatically detects duplicate reports by converting stack traces of function calls to traces of packages. This method, based on trace abstraction, reduces execution time by 93% compared to functions using 1-grams and up to 70% using 2-grams. Future research will focus on package distance and industrial systems. [14] This paper presents the first attempt at using a string matching approach for duplicate bug report detection, surpassing the recall rates of current research approaches by at least 5% using the Firefox bug report repository. The study shows that longest common subsequences and bioinformatics algorithms can effectively detect duplicate reports. The authors also emphasize the importance of proper primary bug report identification and the role of natural language in detecting duplicates. They suggest that application-specific dictionaries could offer new types of analyses that could improve report matching processes, normalizing individual reports and increasing the effectiveness of duplicate report detection and classification algorithms.

III. PROPOSED WORK

In the proposed work, we introduce two distinct utilities, each designed to address specific challenges in bug management. Let's delve into each of them individually, exploring their methodologies in detail. The system architecture diagram as described in Figure 1 consists of 2 parts:

1) On the website's homepage, users can choose if they forest want to categorize defects/ find redundancy in defects.

2) If they want to classify defects they will be ased to upload an excel file, which will be processed, and a new excel file will be generated where 2 new columns will be added, one consisting of the features which have the defects and whether they are UI/Functional.
3) In redundancy detection, they shall upload 2 excel files, one is our original database consisting of all the defcts and the other is the one in which we want to find whether there are any redundant defects.

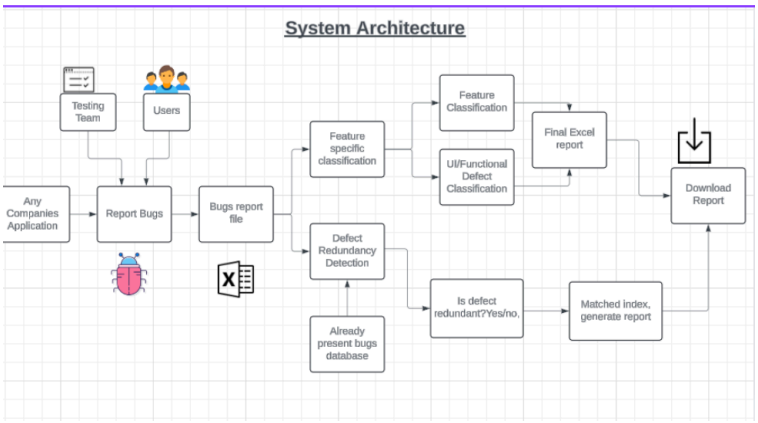


Figure 1. System Architecture

1. Bugs Classification : Bug reports submitted by users and the testing team are stored in a single-column dataset titled "Issue Summary," detailing the nature of the reported bugs. The classification process involves two distinct categories:

- **Category 1**: Identifies the software feature associated with the reported bug. For instance, Table 1 illustrates an example

Issue Summary	Category 1
Button size is not proper on the Login Page.	Login Page
Keypad should be numeric while entering mobile number while doing top up	Keypad

Table 1. Example of category 1

- **Category 2**: Offers a broader classification approach, comprising two classes:
    - *UI :* Encompasses bugs linked to the software's user interface components. Bugs arising from UI elements are classified under this category.
    - *Functional :* Includes bugs related to the functional aspects of the software. Bugs stemming from computational processes fall into this category.

Issue Summary	Category 2
Button size is not proper on the Login Page.	UI
Keypad should be numeric while entering mobile number while doing top up	Functional

Table 2. Example of Category 2 Classification

The working of the proposed solution is represented in FIGURE 1. The system comprises of following units :

- *CSV Conversion:* This module facilitates the transformation of the report file into CSV format, ensuring compatibility with backend operations such as dataframe creation and manipulation.

- *Prepprocessing:* Its main objective is to streamline the representation of terms within bug reports and eliminate irrelevant terms, optimizing the data for further analysis and classification.

- *SGD Classifier:* Utilizing techniques like CountVectorizer, TfidfTransformer, and a classifier, this module analyzes the preprocessed data to classify bug reports into relevant categories efficiently.

- *Report Generation:* Once the bug reports are classified and stored in a dataframe with an additional category column, this module generates a comprehensive report using the structured data, providing insights and summaries for further action.

A. Pre-Processing : Following operations are used to preprocess the textual data of the Issue Summary extracted from bug reports:

- *HTML Decoding:* Removes any HTML tags present in the text using the `BeautifulSoup` library. This ensures the text is in a readable format and removes HTML-related noise.

- *Lowercasing:* Converts all text to lowercase. This standardizes the text and ensures consistency in subsequent processing steps.

- *Symbol Replacement:* Replaces certain symbols with spaces. This helps separate words and aids in subsequent tokenization.

- *Symbol Deletion:* Removes symbols that are not alphanumeric characters or spaces. This eliminates special characters, punctuation marks, or symbols that may not contribute meaningfully to the analysis.

- *Stopword Removal:* Removes stopwords, which are commonly occurring words like "the", "is", "and", etc. This reduces noise in the text and focuses on the more meaningful content.

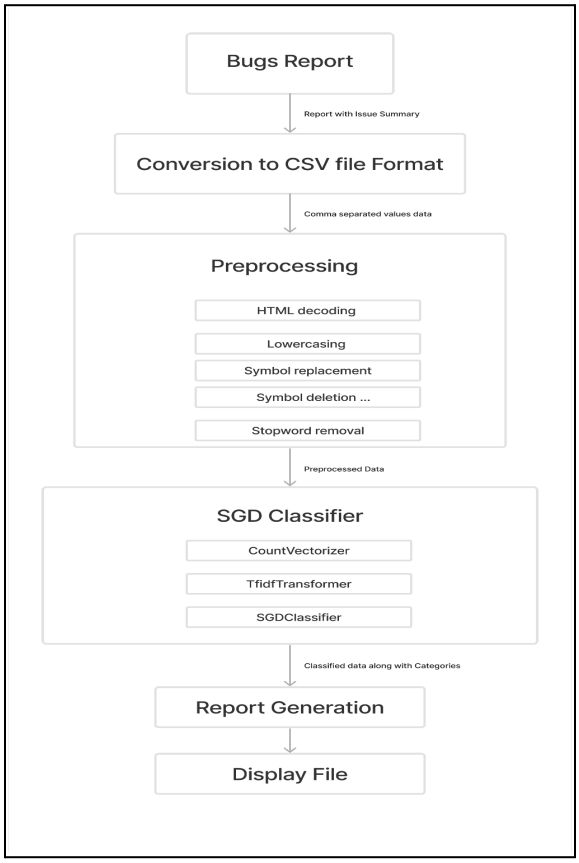


Figure2. End to end flow of algorithm

B. SGD Classifier: One of the most popular optimization strategies for training machine learning models is gradient descent, also known as steepest descent, which minimizes the discrepancy between expected and actual results.The Gradient Descent technique serves as the cornerstone for methods to machine learning and deep learning. SGD, or stochastic gradient descent, is a straightforward method that is incredibly effective for learning linear classifiers with discriminative properties under convex loss functions, including logistic regression and (linear) support vector machines.

- *Count Vectorizer :* In this layer, the textual data undergoes a crucial transformation, converting words into numerical representations. This process involves tokenization, where the text is split into individual words or tokens, followed by counting the occurrences

of each token. Subsequently, a matrix is constructed, where each row corresponds to a document, and each column represents a token. The cell values denote the frequency of each token within each document, forming what is commonly referred to as the "document-term matrix." To further enhance our analysis, we utilized the CountVectorizer method on our dataset. This approach allows us to subsequently apply the TfidfTransformer, enabling us to assess the significance of each token independently. Figure 2 illustrates the document-term matrix generated for the first 50 tuples of our dataset.

```
[[0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 2 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0]
 [0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 1 1]
 [0 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0]]
```

Figure 3. Count Vectorizer Matrix

- *TfidfVectorizer* **:** The TfidfVectorizer, short for "Term Frequency-Inverse Document Frequency Vectorizer," expands upon the capabilities of CountVectorizer by integrating the TF-IDF weighting scheme. TF-IDF, a numerical metric, evaluates the significance of a term (or token) in a document relative to a broader corpus.

The TF-IDF value for a term in a document is computed by multiplying two components:

- *Term Frequency (tf):* This quantifies the frequency of a term within a document. Typically, it's calculated as the count of the term in the document divided by the total number of terms in the document.
- *Inverse Document Frequency (idf):* This assesses the rarity of a term across the entire corpus. It's determined as the logarithm of the total number of documents divided by the number of documents containing the term.

By combining these two components, TF-IDF provides a balanced measure of term importance, considering both the local significance within a document and the global rarity across the corpus. This approach enables TfidfVectorizer to generate feature vectors that emphasize terms that are both frequent within individual documents and distinctive across the entire corpus.

Let's try to analyze the tool's underlying maths.Suppose that we have *t1, t2, t3...tn* bugs summary: *d* is the document, *D* is the set of documents,then Tfidf is given by:

$$tfidf(t,d,D) = tf(t,d) * idf(t,D)$$

The *tf(t,d)* term answers the question, how important is this word to this specific document , and is given by :

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

idf(t,D) quantifies the rarity of a word within a document set D, indicating how common the word is across all documents. It is calculated as:

$$idf(t,D) = log\frac{|D|}{|\{d \in D : t \in d\}|}$$

	app	bar	NBPL buttons	buy	Credit
0	0.0000	0.0000	0.0000	0.4821	0.48214
1	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.3015	0.0000	0.0000	0.0000	0.0000
3	0.0000	0.0000	0.0000	0.0000	0.0000
4	0.0000	0.3983	0.9383	0.0000	0.00000

Table 3. TF-IDF Values

Table 3 displays the values for the first 5 tuples of the dataset. A value of 1 indicates that the term appears in all documents. This information is invaluable for our classification model, as it facilitates classification based on the importance of words.

3. Classifier Model : The way the Classifier tool SGD operates is by gradually adjusting model parameters in the direction of the sharpest gradient of a loss function.The classifier receives the TF-IDF matrix as an input. In order to handle datasets efficiently, the classifier uses a portion of training data to update the model weights during the training process. The algorithm is given as:

Algorithm :SGD

1: initialization
2: input z0 Rd
3: for i=0 to T-1 do
4: uniformly randomly pick i from 1.....n
5: while not converged do do
6: for i= shuffle(1,2.....N) dodo
7: z0+ λ Ji (z) return z
8: return best possible solution

Working : First, the model parameters *(θ)* and learning rate *(α)* are chosen. The training data is then mixed up to introduce some randomness into the dataset.The approach evaluates a single training sample and calculates the cost function gradient *(J)* in relation to the model parameters for each iteration.His gradient illustrates the slope's magnitude and direction.The model makes predictions in the opposite direction of the gradient while attempting to minimize the cost function.

Let *J(θ)* he the cost function, *hθ(x)* be the predicted value and *y* be the actual target,the cost function is given by:

$$J() = \frac{1}{2m} * \sum_{i=1}^{m} (h(x^i) - y^i)^2$$

Gradient is given as :

$$J() = \frac{1}{m} * \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_i^i for \quad j = 0 \rightarrow n$$

2. Duplicate Detection : The duplicate detection utility requires two files: the new bug report and the old report. The user must submit both files; once uploaded successfully, the backend server preprocesses the text by tokenizing, deleting stop words, and extracting important words. Then each tuple of old data and new data tokens is compared; if the comparison score exceeds 70%, the tuple is designated as duplicate and attached to the new data frame, which includes Issue Summary and Duplicate(Yes or No). The system architecture for this feature is given below in Figure 5. Figure 6 and Figure 7 show the pictures of the actual utility developed and comparison of files taken for sample usage. The system also shows at which index did we find a similar defect already reported in the old file.

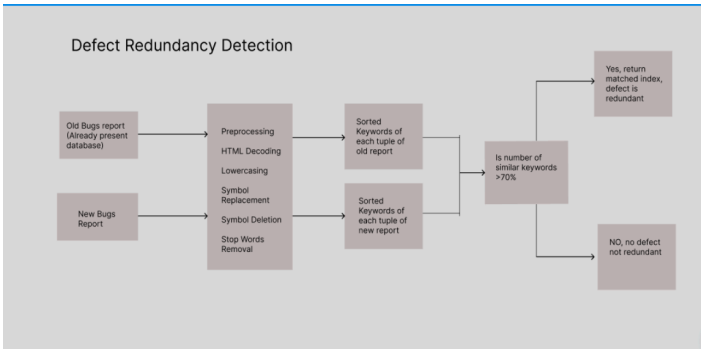


Figure 5. Working of Data Redundancy Detection

Sentence	Duplicate	Matched_Index
Buttons size are not proper on Login Page.	Yes	256
Top label Spacing margin - Make a payment Page	Yes	257
[Mobile App] Keypad should be numeric when user enters mobile number while doing Topup.	Yes	258
Incorrect Monetary value is getting displayed on Buy supplies screen.	Yes	259

Figure 6. Duplicate Detection Utility

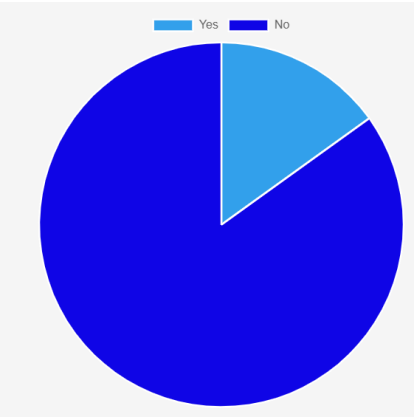
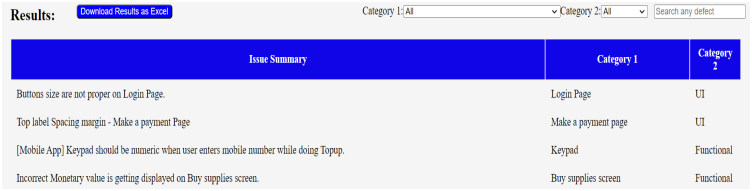


Figure 7. Pie Chart showing redundant defects percentage

IV. RESULTS & CONCLUSION

The system exploited the capabilities of ML algorithms to make the job of software programmers easier. With our first utility, which used an SGD classifier, we were able to reach an accuracy of **75.93% on Category 1 (Figure 9)** and approximately **90.02% on Category 2 (Figure10)**. We also tested the dataset on an intentionally constructed dataset, and the model did exceptionally well on it. Figure 8 shows a output of the classification utility developed for category 1 and category 2.



Issue Summary	Category 1	Category 2
Buttons size are not proper on Login Page.	Login Page	UI
Top label Spacing margin - Make a payment Page	Make a payment page	UI
[Mobile App] Keypad should be numeric when user enters mobile number while doing Topup.	Keypad	Functional
Incorrect Monetary value is getting displayed on Buy supplies screen.	Buy supplies screen	Functional

Figure 8. Classification utility

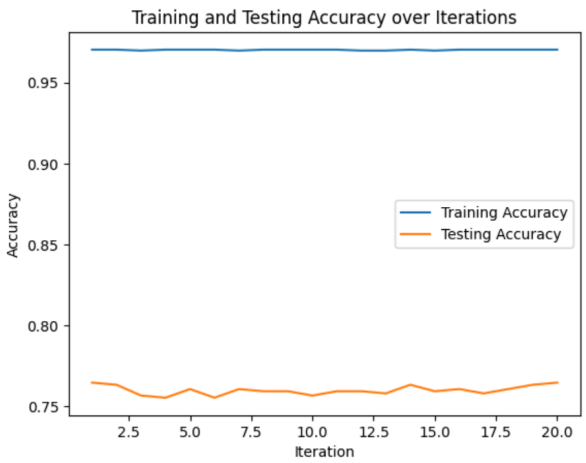


Figure 9. Graph of Training and Testing accuracies for Category 1.

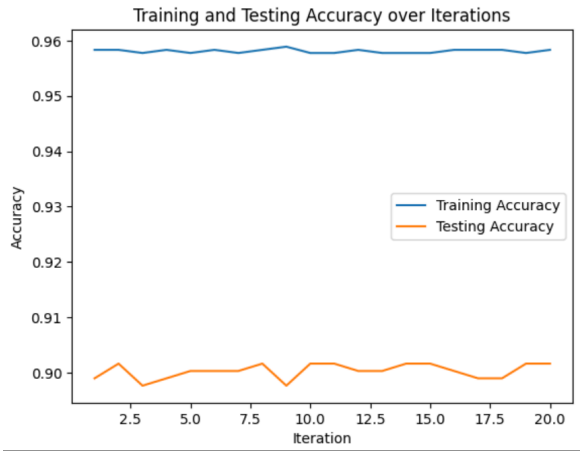


Figure 10. Graph of Training and Testing Accuracies for Category 2 Classification.

The utility can save users a lot of time by providing an analytical view of the features causing the issues, allowing them to work on them more simply. An example of the visualization is shown in Figure 7. The system can be very helpful in quality assurance sections in various where the engineers have to deal with bugs manually and thus can help reduce cost and efforts.

References

[1] D. C. Das and M. R. Rahman, "Security and Performance Bug Reports Identification with Class-Imbalance Sampling and Feature Selection," 2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR), Kitakyushu, Japan, 2018, pp. 316-321, doi: 10.1109/ICIEV.2018.8641045. keywords: {Computer bugs;Security;Feature extraction;Software;Performance evaluation;Mutual information;Bug Report Classification;Security Bug;Performance Bug;Class-Imbalance Learning;Feature Selection},

[2] X. Ye, F. Fang, J. Wu, R. Bunescu and C. Liu, "Bug Report Classification Using LSTM Architecture for More Accurate Software Defect Locating," 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Orlando, FL, USA, 2018, pp. 1438-1445, doi: 10.1109/ICMLA.2018.00234. keywords: {Computer bugs;Training;Multilayer perceptrons;Mathematical model;Software;Logic gates;Recurrent neural networks;Long short-term memory, convolutional neural network, bug locating, bug report},

[3] H. Tahir, S. U. R. Khan and S. S. Ali, "LCBPA: An Enhanced Deep Neural Network-Oriented Bug Prioritization and Assignment Technique Using Content-Based Filtering," in IEEE Access, vol. 9, pp. 92798-92814, 2021, doi: 10.1109/ACCESS.2021.3093170.

[4] A. Kukkar, R. Mohana, Y. Kumar, A. Nayyar, M. Bilal and K. -S. Kwak, "Duplicate Bug Report Detection and Classification System Based on Deep Learning Technique," in *IEEE Access*, vol. 8, pp. 200749-200763, 2020, doi: 10.1109/ACCESS.2020.3033045.

[5]S. Shaikh, L. Changan, M. R. Malik and M. A. Khan, "Software Defect-Prone Classification using Machine Learning: A Virtual Classification Study between LibSVM & LibLinear," 2019 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS), Karachi, Pakistan, 2019, pp. 1-6, doi: 10.1109/MACS48846.2019.9024799.

[6] N. Kanti-Singha Roy and B. Rossi, "Towards an Improvement of Bug Severity Classification," 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, Verona, Italy, 2014, pp. 269-276, doi: 10.1109/SEAA.2014.51.

[7] T. Hirsch and B. Hofer, "Root cause prediction based on bug reports," 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Coimbra, Portugal, 2020, pp. 171-176, doi: 10.1109/ISSREW51248.2020.00067.

[8] T. Akilan, D. Shah, N. Patel and R. Mehta, "Fast Detection of Duplicate Bug Reports using LDA-based Topic Modeling and Classification," 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Toronto, ON, Canada, 2020, pp. 1622-1629, doi: 10.1109/SMC42975.2020.9283289.

[9] K. K. Chaturvedi and V. B. Singh, "Determining Bug severity using machine learning techniques," 2012 CSI SixthInternational Conference on Software Engineering (CONSEG), Indore, India, 2012, pp. 1-6, doi: 10.1109/CONSEG.2012.6349519.

[10].S. Liu, H. Tao and S. Feng, "Text Classification Research Based on Bert Model and Bayesian Network," 2019 Chinese Automation Congress (CAC), Hangzhou, China, 2019, pp. 5842-5846, doi: 10.1109/CAC48633.2019.8996183. keywords: {Bayes methods;Text categorization;Task analysis;Classification algorithms;Machine learning;Machine learning algorithms;Predictive models;Bert model;Bayesian network;text classification},

[11] K. Pan, S. Kim and E. J. Whitehead, Jr., "Bug Classification Using Program Slicing Metrics," 2006 Sixth IEEE International Workshop on Source Code Analysis and Manipulation, Philadelphia, PA, USA, 2006, pp. 31-42, doi: 10.1109/SCAM.2006.6.

[12] K. Aggarwal, T. Rutgers, F. Timbers, A. Hindle, R. Greiner and E. Stroulia, "Detecting duplicate bug reports with software engineering domain knowledge," 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Montreal, QC, Canada, 2015, pp. 211-220, doi: 10.1109/SANER.2015.7081831.

[13] K. K. Sabor, A. Hamou-Lhadj and A. Larsson, "DURFEX: A Feature Extraction Technique for Efficient Detection of Duplicate Bug Reports," 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), Prague, Czech Republic, 2017, pp. 240-250, doi: 10.1109/QRS.2017.35.

[14] S. Banerjee, B. Cukic and D. Adjeroh, "Automated Duplicate Bug Report Classification Using Subsequence Matching,"2012 IEEE 14th International Symposium on High-Assurance Systems Engineering, Omaha, NE, USA, 2012, pp. 74-81, doi: 10.1109/HASE.2012.38.