DOI: https://doi.org/10.52756/ijerr.2024.v45spl.005 *Int. J. Exp. Res. Rev.*, Vol. 45: 56-69 (2024)

Original Article | Peer Reviewed | Open Access

International Journal of Experimental Research and Review (IJERR)
© Copyright by International Academic Publishing House (IAPH)
ISSN: 2455-4855 (Online)
www.iaph.in

ISSN 2455-4855

User Interface Bug Classification Model Using ML and NLP Techniques: A Comparative Performance Analysis of ML Models

Sara Khan* and Saurabh Pal

Department of Computer Applications, Veer Bahadur Singh Purvanchal University, Jaunpur- 222003, India

E-mail/Orcid Id:

SK, computerappl13@gmail.com, https://orcid.org/0000-0002-2614-3929;
SP, drsaurabhpal@yahoo.co.in, https://orcid.org/0000-0001-9545-7481

Article History:

Received: 14th May., 2024
Accepted: 23rd Oct., 2024
Published: 30th Nov., 2024

Keywords:

Bug classification, feature extraction, hyperparameter tuning, imbalance classification, UI

How to cite this Article:
Sara Khan and Saurabh Pal (2024). User Interface Bug Classification Model Using ML and NLP Techniques: A Comparative Performance Analysis of ML Models. *International Journal of Experimental Research and Review*, *45*, 56-69.
DOI:
https://doi.org/10.52756/ijerr.2024.v45spl.005

Abstract: Analyzing user interface (UI) bugs is an important step taken by testers and developers to assess the usability of the software product. UI bug classification helps in understanding the nature and cause of software failures. Manually classifying thousands of bugs is an inefficient and tedious job for both testers and developers. Objective of this research is to develop a classification model for the User Interface (UI) related bugs using supervised Machine Learning (ML) algorithms and Natural Language Processing (NLP) techniques. Also, to assess the effect of different sampling and feature vectorization techniques on the performance of ML algorithms. Classification is based upon 'Summary' feature of the bug report and utilizes six classifiers i.e., Gaussian Naïve Bayes (GNB), Multinomial Naïve Bayes (MNB), Logistic Regression (LR), Support Vector Machines (SVM), Random Forest (RF) and Gradient Boosting (GB). Dataset obtained is vectored using two vectorization techniques of NLP i.e., Bag of Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF). ML models are trained after vectorization and data balancing. The models ' hyperparameter tuning (HT) has also been done using the grid search approach to improve their efficacy. This work provides a comparative performance analysis of ML techniques using Accuracy, Precision, Recall and F1 Score. Performance results showed that a UI bug classification model can be built by training a tuned SVM classifier using TF-IDF and SMOTE (Synthetic Minority Oversampling Techniques). SVM classifier provided the highest performance measure with Accuracy: 0.88, Precision: 0.86, Recall: 0.85 and F1: 0.85. Result also inferred that the performance of ML algorithms with TF-IDF is better than BoW in most cases. This work provides classification of bugs that are related to only the user interface. Also, the effect of two different feature extraction techniques and sampling techniques on algorithms were analyzed, adding novelty to the research work.

Introduction

User Interface Testing, generally abbreviated as 'UI Testing', deals with structurally and functionally testing user interface components. Validation is needed for cross-browser checks, working of dynamic contents, anti-aliasing checks, in-built support plugins and many more (Aho and Vos, 2018). Automating these activities involves thousands of test cases to be created and maintained. Maximum time spent by testers is in analyzing failures of any of these test cases. Analysis of failures starts with bug report analysis and then moves towards fixing bugs and simultaneously towards fixing broken tests (Fazzini et al., 2018). These reports are generally written by stakeholders like end users, developers, testers and project managers who are directly or indirectly associated with project development and maintenance. A typical bug report consists of issues reported in a simple natural language focusing on an unexpected behavior, change request and other supplementary information like logs and screenshots (Vito et al., 2024). One of the major causes of bug report analysis is to find test case failure causes and is one of the important responsibilities of a developer. Bug reports are useful in reproducing failures to avoid similar faults in the future (Kang et al., 2024). Bug reports also help highlight the low test coverage area. Since every UI page has hundreds of UI elements, this bug report helps gather information on expected and unexpected issues. Bugs represent uncertainty in the area and hence become the classification area (Colavito et al., 2024). This research work classifies the bugs into two classes i.e., 'Defect' and 'Enhancement', based upon the 'Summary' feature of the bug report, which is a textual feature. UI bug classification can be represented as equation 1.

$UIB = \{UIB_1, UIB_2, \ldots, UIB_n\}$ and $UIB_x € \{0, 1\}$ (1)

Where, UIB represents a set of UI labeled bugs

n represents total retrieved bugs and, x represents an individual bug.

Label 0 represents 'Defect' and 1 for 'Enhancement' (Antoniol et al., 2008). Classifying the bugs into 'Defect' and 'Enhancement' helps the developer and tester count the defects for quality assurance, identify the area of regression testing, initiate a change management cycle, create new test cases for updates, etc. Bug classification is one of the applications in the field of 'Text Classification' (Meng et al., 2022). Bug report is a combination of structured as well as unstructured information. Structured information includes severity, priority, component, etc. Unstructured information describes or summarizes the bug (Kukkar et al., 2018). 'Summary' feature of the bug helps in understanding the problem and thereafter helps in reproducing the bug if required. From the developer's perspective, issue description, bug reproducing steps and stack traces are the three most important features of the bug report that help in debugging the bug, while other features in the bug report directly impact the bug resolution time (Soltani et al., 2020). Hence, to work towards this problem statement, the research objectives of this study can be stated as below-

# To generate a ML model for automatic classification of the bugs from UI bug repositories into Defect and Enhancement class.

# To compare the performance of classifiers w.r.t feature extraction techniques and sampling techniques on the imbalance dataset.

# To achieve the best-performing classifier for UI bug classification.

Related work

Different approaches have been studied and proposed in the past related to bug categorization, localization, assessment and classification. Since manual classification is time-consuming and error-prone, many approaches have tackled the bug classification problem (Iqbal et al., 2020). This section discusses significant studies related to automated bug analysis and classification. Authors performed bug classification focusing on cloud computing applications using ML and NLP techniques (Tabassum et al., 2023). Four classifiers were evaluated and compared i.e., MNB, LR, Decision Tree (DT) and RF. For vectorization, TF-IDF and Word2Vec extraction techniques were used. Results showed that RF attained the highest testing accuracy of 91.73% and 100% training accuracy. Another study by (Bhandari and Rodriguez-Perez,2023) focused on assessing the performance of various NLP and ML techniques in the classification of intrinsic bugs. They utilized two embedding techniques seBERT (Bidirectional Encoder Representations from Transformers) and TF-IDF, on SVM, LR, DT, RF and K-Nearest Neighbour (KNN) classifiers. Results showed that DT combined with TF-IDF achieved a F1 score of 78% on bug titles. Research work by (Alsaedi et al., 2023) proposes a classification model that can predict the nature of the bug report using ensemble ML techniques. Results highlighted that the proposed approach attained 96.72% accuracy with text augmentation. Authors have provided an approach for using unstructured bilingual reports (Köksal and Tekinerdogan, 2022). Research studied the effect on performance of ML algorithms due to preprocessing, BoW size, K-fold validation and word embeddings. No major difference was seen in the performance results of algorithms related to preprocessing and BoW size of 265. 10-fold cross-validation showed better results in SVM using a linear kernel. Word embedding 'FastText' with RF gave 71.19% of F1- measure. Another paper tried to solve the problem of software bug prediction using ML approach (Hammouri et al., 2018). The author's approach was based on historical data faults, essential metrics, and different software computing techniques. NB, DT and Artificial Neural Network (ANN) classifiers have been used to perform the task. On the three dataset used, DT outperforms the other two techniques with 97.10% accuracy, 99.60% precision measure and 100% recall measure. Authors tried to automate the Orthogonal Defect Classification (ODC) attributes into classes internal, external 1 and external 2 using KNN, RF, Nearest Centroid (NC), NB, SVM and Recurrent Neural Network (RNN) (Lopes et al., 2020). Results showed that there are certain difficulties in automating the process, which can be improved by using bigger datasets. Several authors have directly related the analysis of bugs to fault or defect prediction approaches. The ML-

Table 1. Overview of the related work.

Reference	Application of classification	ML techniques applied	Classification labels	Result
Tabassum et al., 2023	Classification of Cloud computing applications	MNB, DT, LR, RF	Crash, Enhancement, Performance, Resource Usage, Security and Compile	RF achieved the highest test accuracy of 91.73%.
Bhandari and Rodríguez-Pérez, 2023	Classification in Version Control Systems	SVM, LR, DT, RF and KNN	Intrinsic, Non-Intrinsic Bugs/ Extrinsic Bugs	TF-IDF with DT achieved highest F1 Score of 78%.
Alsaedi et al., 2023	Classification of Mozilla and Eclipse bug repositories	RF, MNB, SVC, LR, Proposed ensemble model (Hard voting) and ensemble model(Soft voting)	Graphical User Interface (GUI), Network or Security, Program Anomaly, Configuration, Test Code and Performance	Proposed model attains accuracy of 96.72% with text augmentation.
Köksal and Tekinerdogan, 2022	Classification of unstructured bilingual bug report	NB, SVM, KNN, LR, DT, RF	Assignment/Initialization, External Interface, Internal Interface, Others	FastText with RF gave highest F1 measure of 71.19%.
Hammouri et al., 2018	Classification of data faults, essential metrics and soft computing techniques	NB, DT, ANN	Fault Categories A(0-4), B(5-9), C(10-14), D(15-19) and E(more than 20)	DT achieves highest accuracy with 97.10% accuracy, 99.60% precision measure and 100% recall measure.
Lopes et al., 2020	Classification of ODC attributes	NB, SVM, NC, KNN, RF, RNN	Internal, External 1, External 2	Results highlight the challenges of automating ODC attributes and opportunity to improve with larger datasets.
Alqahtani, 2023	Classification of Security bug reports	SVC, RF, LR, GNB, KNN, Multi-Layer Perceptron (MLP)	Security and Non-security bugs	Fasttext classifier achieves F1 score of 0.81
Goseva-Popstojanovaa nd Tyo, 2018	Classification on NASA datasets	Supervised: Bayesian network, KNN, NB, MNB, RF, SVM Unsupervised: Cosine similarity distance measure	Supervised: Security and Non-Security bugs Unsupervised: Non-Security bugs	Both can be used for classification. Supervised learning performed slightly better
Hirsch, and Hofer, 2022	Classification on bug reports of GitHub projects	MNB, SVM, RF, LR, ensemble classifiers	Concurrency, Memory and Resources, Other, Semantic	Ensemble classifier performance was better than single classifier alone

driven bug classification approach has been applied in several ways to predict software defects. In (Alqahtani, 2023), security bug classification has been done using the ‘FastText’ classifier. Results highlighted that 0.81 F1 score was achieved in identifying security bug reports and 0.61 in cross project validation. Authors of paper (Goseva-Popstojanova and Tyo, 2018) performed automated bug classification related to security and non-security issues, using both supervised and unsupervised algorithms. Results showed that supervised algorithms performed better than unsupervised algorithms. In (Hirsch and Hofer, 2022), a classification model has been proposed based on ensemble methods to predict the fault category. Comparison of basic ML algorithms with ensemble methods has been done. Result obtained is 0.69% macro average F1 score. It was also applied to inter projects for validation.

Studying and analyzing the available literature, it was found that classification specific to UI issues has been rarely done. Classifying these issues or bugs can help in risk profiling and can help in improving risk coverage in UI test suites, which will further help in organizing the test activities. Table 1 provides a short summary of the related work highlighting application of the classification being done, ML techniques applied, labels of the classification and major result obtained. Rest of the paper is structured as follows. Section II elaborates on materials and methods, section III describes results and discussion and section IV provides the conclusion.

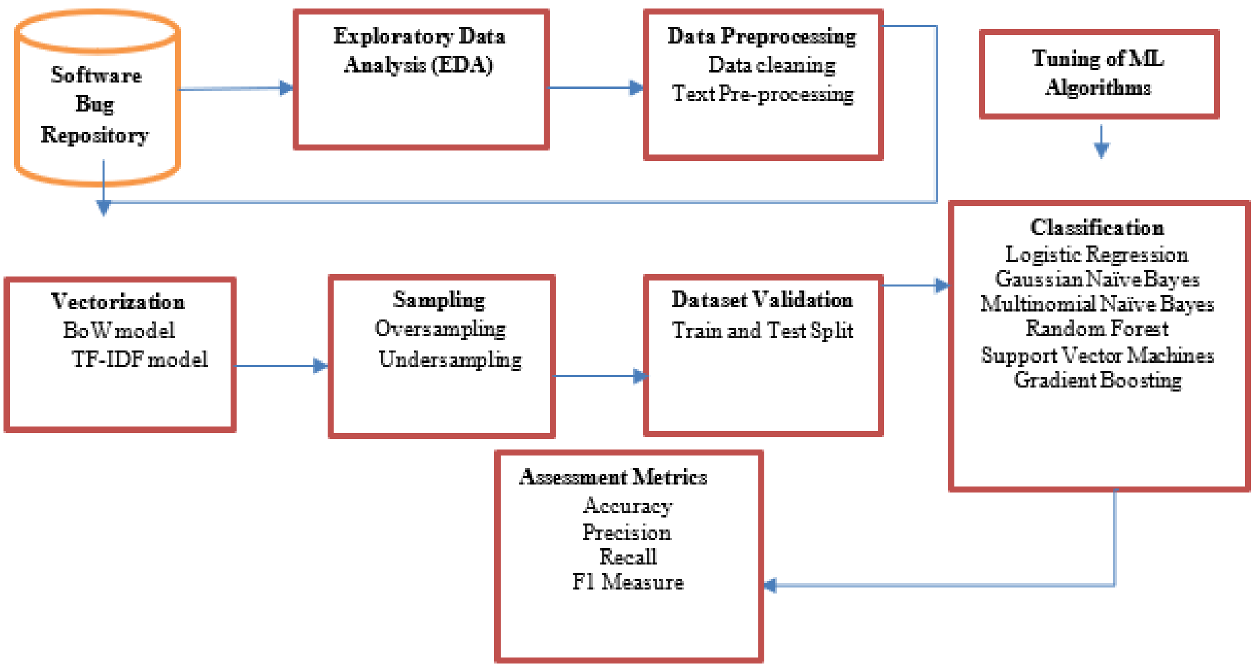

Figure 1. Methodology of the study.

Table 2. Components for which UI issues have been retrieved from the bug repository.

Sl. No	Components	Tool name
1	Accessibility Tools	DevTools
2	API	Testopia, Mozilla
3	DOM	DevTools
4	Frontend	Web Extensions
5	Graphic Design	Mozilla Foundation Communications
6	Graphics	Core
7	Layout	Core
8	Pages & Content	Mozilla
9	Panning & Zooming	Core
10	Picture-in-Picture	Toolkit
11	Shared Components	DevTools
12	Tabs	Fenix
13	Theme & Visual Design	Firefox for iOS
14	UI	Taskcluster
15	UI Design	Seamonkey
16	User Interface	Bugzilla, Testopia, Mozilla
17	Widget	Core

Materials and Methods

Methodology

Figure 1 provides high level design of the system model. It comprises of six phases: 1. Data Retrieval and Analysis 2. Data Pre-processing 3. Vectorization 4. Class Balancing 5. ML classification and 6. Performance assessment. In the first phase, dataset has been retrieved from the bug repository and analyzed. Further in the second phase, it is cleaned for null and incomplete values using NLP text pre-processing techniques. Third phase is about selecting the relevant feature and further transforming it into a model processing format using NLP vectorization techniques. In the fourth phase class balancing techniques are applied to remove imbalance in the dataset. Dataset is further split in 70:30 mode of train-test split for training and validating the ML model. In the fifth phase, optimization and training of the classifiers have been done on the feature vectors. Finally, the model performance is evaluated and compared in the sixth phase. Detail description about each phase is provided below-

About the dataset

Gathering quality data is considered the most crucial step in building a classification model that can have a significant impact on the performance of ML models. Therefore, it was assured that data collected is of high quality and free from irrelevant information. Dataset has been retrieved from the Eclipse Bugzilla repository (Ahmed et al., 2021). It is a very powerful web based system used in maintaining thousands of bug reports. It provides an efficient way to track the bugs. Around 10K issues were filtered on the basis of different UI components. Table 2 provides components for which issues have been extracted. And, Table 3 provides bug report attributes.

Dataset has two classes i.e., ‘Defect’ and ‘Enhancement’ as mentioned in the previous section. After applying data cleaning techniques like removal of null and incomplete values. Distribution of these two classes that was obtained is represented in Figure 2. Around 10K issues were initially extracted, out of which around 200 issues were dropped from the dataset after cleaning due to inappropriate attribute values. It can be seen that the dataset obtained is highly imbalanced. To balance the dataset, different sampling techniques have been applied which are mentioned in the next section.

Table 3. Attributes of the bug report.

Sl. No	Attribute Name	Description
1	Bug ID	Unique bug id number to track throughout the cycle
2	Type	Classification of bugs as ‘Defect’ or ‘Enhancement’
3	Summary	Detail description of issue
4	Product	Product name associated
5	Component	UI component associated
6	Assignee	Assignee name
7	Status	New, Assigned, Unconfirmed, Closed
8	Severity	Normal/Minor/Major
9	Priority	P1/P2/P3
10	Version	Version number of the product

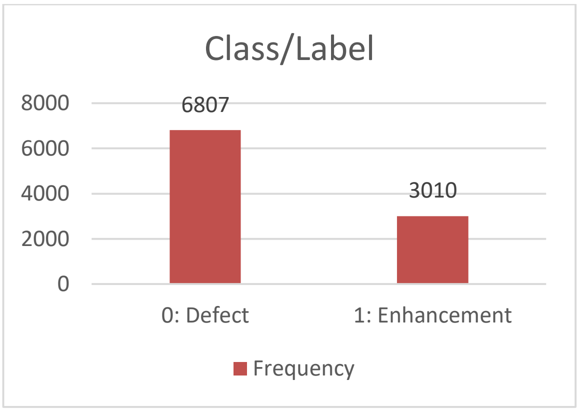

Figure 2. Distribution of classes.

A sample of the UI bug reports from the dataset

Table 4. Bug sample from the dataset representing Defect and Enhancement.

Type	Summary
Defect	The icons normally displayed by Testopia are predominantly absent
Defect	Environment property values are not saved
Enhancement	Show colour contrast ratio between icon image and its background
Enhancement	Graphics data gathering from about: support is slow

representing both the classes is provided in Table 4.

Classification using ML

Classification is the process of assigning a class to the object by prediction. Each class is represented by some features that are similar in the behavior or in pattern. These features contain adequate information that can distinguish one class from another. Following are the classifiers applied in this research work.

Logistic Regression Classifier

Generally abbreviated as ‘LR’. LR calculates the probability of an object to be assigned to a particular class. An estimated probability greater than 50% indicates that an instance belongs to the positive class which is labeled as 1 and 0 otherwise. A weighted sum of the input features along with the bias term is calculated by the model and the corresponding output is further provided (Starbuck, 2023).

Support Vector Machines classifier

Generally abbreviated as ‘SVM’. Objective of the SVM classifier is to find the hyperplane which tries to maximize the margin between the classes in training data. Advantage of SVM is that it utilizes very little memory and is good with high dimensional complex data. They are generally affected by points which are near the margin of the hyperplane. Integrating SVM with different kernel methods makes the classifier adaptable to different types of data (Steidl, 2015).

Naïve Bayes classifier

Based on the Bayes’ theorem of probability, this algorithm produces the probabilities for every case. It predicts the highest probability outcome. Assumption is based on the fact that features are independent. In case of text classification, a text vector is constructed of corresponding document x, depending on a given dictionary, wherein, n represents total elements in the dictionary set DS and total occurrences in document x (Li et al., 2022).

DS= {ds1, ds2,............, ds_n}

X= {x1, x2,.............,x_n}

Objective is to find the maximum probability that can solve equation ‘2’ given below.

$$P(y_j \mid X) = [P(y_j)P(X \mid y_j)]/P(X) \qquad (2)$$

Any new data set X when provided, there will be calculation for all P (y_j|X), and the greatest probability value will be the classification result of the document X which will belong to classification class y. In this research work, GNB and its variation MNB classifiers have been used.

Random Forest

Generally abbreviated as ‘RF’. This is one of the most well accepted ensemble methods which are based on the bagging methods. It is a technique that ensembles decision trees together. Voting methods based on the output of the results of individual trees determine the classification results. RF brings randomness when growing the trees. As in the case of normal decision trees, which finds best feature for splitting the node. RF finds the best feature from the random set of features. This randomness makes the tree diverse with lower variance, making it a better model (Paul et al., 2018).

Gradient Boosting

Generally abbreviated as ‘GB’. This is a boosting algorithm that works by adding predictors sequentially to an ensemble. New predictors correct the predecessor. At every stage regression trees try to fit on the log loss made by the previous predictor (Chen and Guestrin, 2016). XGBoost python library is used that provide optimized implementation of Gradient Boosting classifiers. Extremely fast, scalable and portable.

Class Balancing Techniques

Classification results can be affected if the dataset is skewed or imbalanced. Classification with an imbalanced dataset will result in misclassification with gradient being less informative. Classes in a dataset that contribute towards a larger proportion of the dataset are called “Majority classes”. And the one that is smaller in proportion is termed as “Minority classes”. There is mainly three categories of imbalance (degree of imbalance) i.e., Mild, Moderate and Extreme. Proportion of minority class is around 20-40% of the dataset in case of the mild, for moderate it’s around 1 to 20% and for extreme this range to less than 1%. There are various important techniques that have been considered for class imbalancing in this research work. Oversampling techniques balances the minority classes by increasing its frequency whereas, undersampling balances the data by lowering the majority class. In the oversampling category, Synthetic Minority Oversampling Techniques (SMOTE), Edited Nearest Neighbour based SMOTE (SMOTE-ENN), SMOTE-Borderline, SMOTE-SVM, RandomOverSampler and Adaptive Synthetic (ADASYN) has been used. In undersampling techniques, RandomUnderSampler, AllKNN, NearMiss and Tomek’s Links have been considered. Details of these techniques can be studied (Hasib et al., 2020).

Hyperparameter Tuning (HT)

HT and optimization is one of the important steps for fine tuning the behavior of ML algorithms. It directly impacts the performance of algorithms. It refers to finding out the set of hyperparameter values that will provide the best performance measures on the data in a reasonable amount of time. It is generally determined by the user before the model is trained. Three kinds of hyperparameter optimization methods are generally used i.e., grid search, manual search and random search. In this research, grid search has been used. As the name itself suggests, in grid search the user predetermines a grid of hyperparameters and the model is trained based on each possible combination of the grid (Ghawi and Pfeffer, 2019).

Text preprocessing

Data cleaning is an integral part of any NLP problem. It is necessary before one could represent the data in a suitable format. Steps of text preprocessing could depend upon the requirement of the problem statement (Hickman et al., 2020). Important preprocessing steps taken in this work are stop words removal, conversion to lowercase, tokenization, stemming and lemmatization.

# Stop words removal: Removing the common words that act as a noise to the statement like 'a', 'the' etc.

# Tokenization: Converting strings to lists of words.

# Stemming: Words that imply the same meaning are identified as the same. This is achieved by removing '--ing','--ly' etc.

# Lemmatization: Converting to the base or root word. For eg. 'feet' becomes 'foot'.

Extracting text features

Features need to be extracted from the text data as it's not in the required form that can be used for ML techniques. Two important methods which are used in this research for feature extraction are BoW and TF-IDF. In the BoW model, each word is considered as a feature and it checks for a word's existence in a sentence. Hence, each sentence represents a BoW. Each sentence can be referred to as a document and collection of all documents can be said as a corpus. Each document is converted to a vector representing words present in the document in a dictionary. Major drawback of BoW is that it does not preserve the order of the sentence but still it's being applied in various classification tasks because of its simplicity. Words present in a document are assessed for its importance by TF-IDF. Less occurring words have a larger significance than more occurring words which have lower significance.

Performance Evaluation Metrics

To measure the performance of the classification model, four powerful metrics have been used. They are briefly described below. Notations used are TP: True Positives, TN: True Negatives, FP: False Positives, and FN: False Negatives (Juba and Le, 2019).

1. Accuracy: Accuracy denotes 'Right predictions'. It can be computed as the amount of correct predictions out of total predictions. Equation 3 provides formula to measure Accuracy.

$$[TP + TN] \big/ [TP + TN + FP + FN] \tag{3}$$

2. Precision: Precision denotes proportion of positive identification that was actually correct. Equation 4 provides formula to measure Precision.

$$[TP] \big/ [TP + FP] \tag{4}$$

3. Recall: Recall represents the proportion of actual positives that were identified correctly. Equation 5 provides formula to measure Recall.

$$[TP] \big/ [TP + FN] \tag{5}$$

4. F1 Measure: Integrates precision and recall for better performance measure. Is calculated by taking harmonic mean of the two. Equation 6 provides formula to measure F1 score.

$$2 \times [Precision \times Recall] \big/ [Precision + Recall] \tag{6}$$

Results and Discussion

Preprocessing Results

Six classifiers namely LR, GNB, MNB, RF, SVM and GB have been trained on the dataset for the classification objectives after required preprocessing and hyperparameter tuning have been applied, as mentioned in the previous section. Results for the preprocessing are provided in Table 5 and 6.

Table 5. Text preprocessing methods applied to the 'Summary' feature of the bug report

Summary	Show or allow to go to the component description…….
Tokenization	Show, or, allow, to, go, to, the, component, description ……..
Lowercase	show, or, allow, to, go, to, the, component, description……
Stop words Removal	show, allow, go, component, description…….
Lemmatisation/Stemming	show, allow, go, component, description…….

After preprocessing, feature extraction has been done using Tf-IDF and BoW techniques as mentioned in the previous section. Below is an example of feature vectorization vocabulary created using TF-IDF technique.

{'show': 3523, 'allow': 180, 'go': 1565, 'compon': 757, 'descript': 947, 'move': 2174, 'edit': 1104, 'bug': 581, 'icon': 1696, 'normal': 2269…….}

Table 6. Final output after text preprocessing.

Label	*Summary*	*Length*	*Summary_after_clean*	*length_after_clean*
Enhancement	Show or allow to go to the component description….	86	show allow go component description moving com..	61
Defect	The icons normally displayed by testopia are predominantly...	66	icon normally displayed testopia predominantly...	53
Defect	Editing environments adding new item to top no...	54	editing environment adding new item top node f...	50
Defect	Environment property values are not saved	41	environment property value saved	32
Defect	Environment properties got all the same propertyid	51	environment property got propertyid	35

Hyperparameter Tuning (HT) of ML algorithms

HT of ML algorithms has been done through grid search technique. Best parameter value has been retrieved and used as provided in Table 7. GridSearchCV provides an exhaustive search for an estimator through a parameter grid, providing the best value to the algorithms (Subramani et al., 2022).

Table 7. Hyperparameter Tuning through GridSearchCV.

ML Model	Hyperparameter values obtained through GridSearchCV	
	TF-IDF	**BoW**
GNB	var_smoothing =3.5e-05	var_smoothing=0.00012
MNB	alpha=0.3	alpha=1.0
LR	C=10	C=0.1
SVM	C=1, gamma=1, kernel=linear	C=0.1, gamma=0.001,kernel=linear
RF	min_samples_split=8, n_estimators=100, max_depth=2, max_features=sqrt, min_samples_leaf=3	min_samples_split=8, n_estimators=100, max_depth= 80, max_features=2, min_samples_leaf=3
GB	learning_rate=0.1, max_depth=3, n_estimators=50	learning_rate=0.1, max_depth=5, n_estimators=50

Performance Evaluation

Objective of this research is to formulate best classification model for UI bugs by comparing different feature vectorization methods, sampling methods, and ML techniques. ML models are trained for respective combinations of sampling and vectorization methods. These models are optimized using grid search approach as mentioned in the previous section. Dataset was split in 70:30 ratio meaning 70% data was applied in training the model and 30% in testing the model. Models are then evaluated depending on the metrics. Though, accuracy is considered one of the most important metrics for evaluation, but in certain cases high value of accuracy not always imply a good performing classifier. Therefore, it is necessary to evaluate the models on all the four metrics (Ramay et al., 2019). In this subsection, performance of each model is presented w.r.t its performance measures in tabular format. Highest values obtained in each of the tables are highlighted with a pink colour. Table 8 provides the performance result of GNB.

Table 8. Performance result of GNB.

Sampling Type	Sampling Methods	Accuracy		Precision		Recall		F1 Score	
		TF-IDF	BoW	TF-IDF	BoW	TF-IDF	BoW	TF-IDF	BoW
OverSampling	RandomOverSampler	0.67	0.62	0.77	0.75	0.67	0.61	0.75	0.67
	ADASYN	0.68	0.73	0.88	0.76	0.77	0.71	0.82	0.73
	Borderline SMOTE	0.68	0.72	0.76	0.76	0.65	0.69	0.70	0.72
	SMOTE	0.68	0.73	0.76	0.74	0.66	0.71	0.70	0.72
	SMOTE-ENN	0.50	0.54	0.79	0.76	0.52	0.55	0.62	0.63
	SMOTE-SVM	0.66	0.72	0.77	0.75	0.67	0.71	0.71	0.73
UnderSampling	RandomUnderSampler	0.53	0.53	0.79	0.77	0.54	0.54	0.64	0.63
	AllKNN	0.65	0.65	0.77	0.75	0.50	0.59	0.57	0.66
	NearMiss	0.70	0.70	0.80	0.74	0.69	0.48	0.74	0.58
	Tomek's Links	0.66	0.66	0.88	0.76	0.67	0.61	0.76	0.67

It can be derived that accuracy score with TF-IDF in all the cases of sampling, lies in between 0.50 to 0.70 and with BoW it ranges in between 0.53 to 0.73. F1 score obtained is at the lower side, mostly ranging in between 0.57 to 0.70 with both the vectorization cases. It can be said by analyzing the performance that the model is underfitting the training data to some extent. Table 9 highlights the performance of MNB.

Table 9. Performance result of MNB.

Sampling Type	Sampling Methods	Accuracy		Precision		Recall		F1 Score	
		TF-IDF	BoW	TF-IDF	BoW	TF-IDF	BoW	TF-IDF	BoW
OverSampling	RandomOverSampler	0.78	0.74	0.82	0.81	0.77	0.73	0.79	0.76
	ADASYN	0.78	0.80	0.69	0.80	0.82	0.78	0.74	0.78
	Borderline SMOTE	0.77	0.80	0.83	0.81	0.77	0.78	0.79	0.79
	SMOTE	0.79	0.81	0.83	0.81	0.78	0.79	0.80	0.79
	SMOTE-ENN	0.25	0.36	0.82	0.80	0.30	0.39	0.43	0.52
	SMOTE-SVM	0.71	0.85	0.77	0.78	0.73	0.81	0.74	0.79
UnderSampling	RandomUnderSampler	0.74	0.69	0.82	0.82	0.73	0.70	0.77	0.75
	AllKNN	0.85	0.77	0.82	0.82	0.82	0.77	0.82	0.79
	NearMiss	0.76	0.51	0.80	0.79	0.75	0.54	0.77	0.64
	Tomek's Links	0.86	0.82	0.85	0.82	0.83	0.81	0.83	0.81

Performance of MNB on the dataset is better when compared to GNB. Accuracy ranges in between 0.75 to 0.80 in most of the cases of TF-IDF and BoW and F1 score is mostly higher than 0.70 with most of the sampling techniques. Table 10 provides the performance results of LR. Results highlight that most of the sampling methods are seen to perform well with LR. Where, BoW vectorization technique is seen to have better overall performance compared to TF-IDF with accuracy mostly ranging above 0.80 and F1 mostly higher than 0.75.

Table 11 highlights performance of the SVM classifier. Oversampling techniques is seen to perform well with TF-IDF technique. Accuracy ranges above 0.85 and F1 score above 0.83 in most of the cases.

Table 12 and Table 13 provides performance result for RF and GB algorithms. RF results are mostly similar to SVM in most of the cases with accuracy and F1 score generally above 0.80. Performance of GB is also good with F1 score ranging in between 0.75 to 0.82.

Analyzing the performance of each classifier w.r.t feature vectorization and sampling techniques from the Tables [8-13], final result of each classifier can be presented as in Table 14 and Figure 3 provides visualisation for the same.

Table 10. Performance result of LR.

Sampling Type	Sampling Methods	Accuracy		Precision		Recall		F1 Score	
		TF-IDF	BoW	TF-IDF	BoW	TF-IDF	BoW	TF-IDF	BoW
OverSampling	RandomOverSampler	0.82	0.81	0.82	0.81	0.80	0.78	0.80	0.79
	ADASYN	0.82	0.80	0.83	0.79	0.81	0.78	0.81	0.78
	Borderline SMOTE	0.82	0.82	0.83	0.80	0.81	0.80	0.81	0.80
	SMOTE	0.81	0.82	0.82	0.80	0.80	0.80	0.80	0.80
	SMOTE-ENN	0.27	0.30	0.82	0.82	0.32	0.35	0.46	0.49
	SMOTE-SVM	0.84	0.81	0.83	0.79	0.82	0.79	0.82	0.79
UnderSampling	RandomUnderSampler	0.73	0.70	0.82	0.81	0.73	0.70	0.77	0.75
	AllKNN	0.84	0.83	0.83	0.81	0.82	0.81	0.82	0.81
	NearMiss	0.70	0.42	0.80	0.77	0.70	0.44	0.74	0.56
	Tomek's Links	0.86	0.87	0.82	0.82	0.83	0.82	0.82	0.82

Table 11. Performance result of SVM.

Sampling Type	Sampling Methods	Accuracy		Precision		Recall		F1 Score	
		TF-IDF	BoW	TF-IDF	BoW	TF-IDF	BoW	TF-IDF	BoW
OverSampling	RandomOverSampler	0.86	0.85	0.83	0.83	0.83	0.83	0.83	0.83
	ADASYN	0.87	0.86	0.83	0.80	0.83	0.82	0.83	0.80
	Borderline SMOTE	0.87	0.85	0.86	0.79	0.84	0.81	0.84	0.79
	SMOTE	0.88	0.84	0.86	0.79	0.85	0.80	0.85	0.79
	SMOTE-ENN	0.41	0.33	0.82	0.78	0.45	0.37	0.58	0.50
	SMOTE-SVM	0.87	0.86	0.84	0.82	0.33	0.82	0.47	0.82
UnderSampling	RandomUnderSampler	0.76	0.15	0.82	0.83	0.76	0.20	0.78	0.32
	AllKNN	0.86	0.85	0.83	0.68	0.83	0.80	0.83	0.73
	NearMiss	0.67	0.76	0.81	0.72	0.68	0.71	0.73	0.71
	Tomek's Links	0.87	0.85	0.85	0.68	0.83	0.80	0.83	0.73

Table 12. Performance result of RF.

Sampling Type	Sampling Methods	Accuracy		Precision		Recall		F1 Score	
		TF-IDF	BoW	TF-IDF	BoW	TF-IDF	BoW	TF-IDF	BoW
OverSampling	RandomOverSampler	0.86	0.48	0.83	0.83	0.83	0.83	0.83	0.83
	ADASYN	0.86	0.64	0.82	0.76	0.83	0.50	0.82	0.60
	Borderline SMOTE	0.86	0.66	0.82	0.73	0.82	0.65	0.82	0.68
	SMOTE	0.87	0.72	0.84	0.74	0.83	0.69	0.83	0.71
	SMOTE-ENN	0.45	0.15	0.82	0.03	0.49	0.20	0.61	0.05
	SMOTE-SVM	0.87	0.72	0.83	0.82	0.83	0.82	0.83	0.82
UnderSampling	RandomUnderSampler	0.58	0.41	0.81	0.79	0.59	0.43	0.68	0.55
	AllKNN	0.84	0.86	0.67	0.69	0.80	0.80	0.72	0.74
	NearMiss	0.56	0.52	0.77	0.74	0.57	0.53	0.65	0.61
	Tomek's Links	0.86	0.84	0.69	0.67	0.80	0.80	0.74	0.72

Table 13. Performance result of GB

Sampling Type	Sampling Methods	Accuracy		Precision		Recall		F1 Score	
		TF-IDF	BoW	TF-IDF	BoW	TF-IDF	BoW	TF-IDF	BoW
OverSampling	RandomOverSampler	0.81	0.76	0.82	0.81	0.79	0.73	0.80	0.76
	ADASYN	0.86	0.74	0.82	0.76	0.82	0.74	0.82	0.75
	Borderline SMOTE	0.86	0.77	0.82	0.78	0.83	0.75	0.82	0.76
	SMOTE	0.84	0.78	0.80	0.79	0.81	0.76	0.80	0.77
	SMOTE-ENN	0.38	0.30	0.82	0.82	0.79	0.35	0.80	0.49
	SMOTE-SVM	0.85	0.79	0.81	0.80	0.82	0.76	0.81	0.77
UnderSampling	RandomUnderSampler	0.82	0.67	0.81	0.82	0.79	0.68	0.79	0.74
	AllKNN	0.87	0.86	0.84	0.80	0.82	0.81	0.82	0.80
	NearMiss	0.53	0.47	0.79	0.75	0.54	0.50	0.64	0.60
	Tomek's Links	0.85	0.85	0.86	0.83	0.80	0.80	0.82	0.81

Table 14. Best combination of techniques for each ML model with highest performance measures.

Best Performing combination of classifier <Model, Vectorization , Sampling>	Accuracy	Precision	Recall	F1
<GNB, TF-IDF, ADASYN>	0.68	0.88	0.77	0.82
<MNB, TF-IDF, Tomek's Links>	0.86	0.85	0.83	0.83
<LR, BoW, Tomek's Links>	0.87	0.82	0.82	0.82
<SVM, TF-IDF, SMOTE>	0.88	0.86	0.85	0.85
<RF, TF-IDF, SMOTE>	0.87	0.84	0.83	0.83
<GB, TF-IDF, AllKNN>	0.87	0.84	0.82	0.82

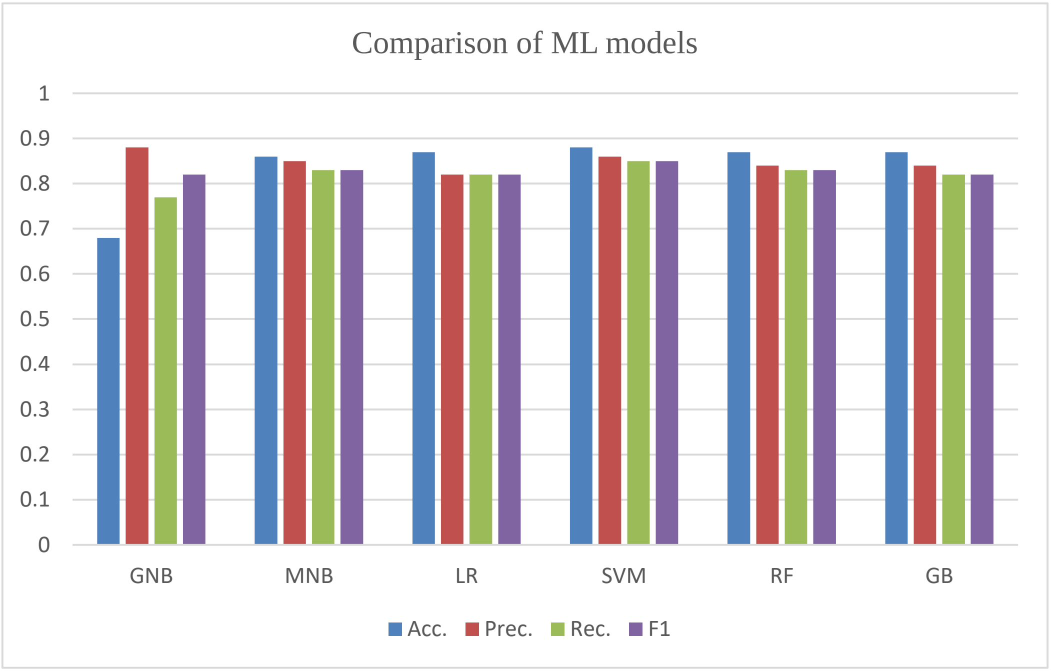

Figure 3. Comparison of ML models with highest performance measure.

So, the best classification model can thus be derived from these analysis. Analyzing Table 14 and Figure 3, it can be deduced that SVM classifier using TF-IDF and SMOTE outperformed all the classifiers, w.r.t all the combinations of sampling and vectorization techniques by achieving 0.88 Accuracy, 0.86 Precision, 0.85 Recall and 0.85 F1 score. Closely followed by RF classifier, when applied with TF-IDF and SMOTE, provides 0.87 Accuracy score, 0.84 Precision score, 0.83 Recall score and 0.83 F1 score.

Conclusion

In this work, we have considered comparing ML algorithms in scenarios of two different vectorization techniques and different oversampling and undersampling data balancing techniques. Two important conclusions can be derived from the results. Hypertuned SVC and RF with TF-IDF and SMOTE techniques almost provided the highest performance measures, with SVM having slightly better overall metrics score. Based on the project feasibility related to run time complexity and resource constraints, a decision to choose between the two can be taken. Second result can be derived that 'TF-IDF' provided better results in most of the cases as compared to BoW in this problem domain. This research work can help in different ways like textual analysis of bug reports through bug classification, pattern analysis, anomaly analysis and many more. This can be a starting step towards automated bug repairing, test suite management and providing continuous improvement to the product.

In future, this work can be extended to analyze the effect of data complexity on the functioning of sampling techniques. More ensemble methods can be applied and results can be compared. Also, more defect repositories can be used with different features. Word embeddings and deep learning techniques can be explored for better performance.

Conflict of Interest

The authors declare that there is no conflict of interest.

References

Ahmed, H.A., Bawany, N.Z., & Shamsi, J.A. (2021). CaPBug-A Framework for Automatic Bug Categorization and Prioritization Using NLP and Machine Learning Algorithms. *IEEE Access*, *9*, 50496-50512. https://doi.org/10.1109/ACCESS.2021.3069248

Aho, P., & Vos, T.E. (2018). Challenges in Automated Testing Through Graphical User Interface. *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 118-121. *https://doi.org/10.1109/icstw.2018.00038*

Alqahtani, S. S. (2023). Security bug reports classification using fast text. *International Journal of Information Security*, *23*(2), 1347–1358. https://doi.org/10.1007/s10207-023-00793-w

Alsaedi, S.A., Noaman, A.Y., Gad-Elrab, A.A., & Eassa, F.E. (2023). Nature-Based Prediction Model of Bug Reports Based on Ensemble Machine Learning Model. *IEEE Access*, *11*, 63916-63931. https://doi.org/10.1109/ACCESS.2023.3288156

Antoniol, G., Ayari, K., Penta, M.D., Khomh, F., & Guéhéneuc, Y. (2008). Is it a bug or an enhancement? : a text-based approach to classify change requests. *Conference of the Centre for Advanced Studies on Collaborative Research*, pp. 304-318. https://doi.org/10.1145/1463788.1463819

Bhandari, P., & Rodríguez-Pérez, G. (2023). BuggIn: Automatic Intrinsic Bugs Classification Model using NLP and ML. *Proceedings of the 19th International Conference on Predictive Models and Data Analytics in Software Engineering*.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). *Association for Computing Machinery*, New York, NY, USA, pp. 785–794. https://doi.org/10.1145/2939672.2939785.

Colavito, G., Lanubile, F., Novielli, N., & Quaranta, L. (2024). Leveraging GPT-like LLMs to Automate Issue Labeling. *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*, pp. 469-480. https://doi.org/10.1145/3643991.3644903

Fazzini, M., Prammer, M., d'Amorim, M., & Orso, A. (2018). Automatically translating bug reports into test cases for mobile apps. *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. https://doi.org/10.1145/3213846.3213869

Ghawi, R. & Pfeffer, J. (2019). Efficient Hyperparameter Tuning with Grid Search for Text Categorization using KNN Approach with BM25 Similarity. *Open Computer Science*, *9*, 160 – 180. https://doi.org/10.1515/comp-2019-0011

Goseva-Popstojanova, K. & Tyo, J. (2018). Identification of Security Related Bug Reports via Text Mining

Using Supervised and Unsupervised Classification. In *Proceedings of IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 344-355. https://doi.org/10.1109/QRS.2018.00047.

Hammouri, A., Hammad, M., Alnabhan, M. M. & Alsarayrah, F. (2018).Software Bug Prediction using Machine Learning Approach. *International Journal of Advanced Computer Science and Applications*, *9*(2). http://dx.doi.org/10.14569/IJACSA.2018.090212.

Hasib, K. M. et al. (2020). A Survey of Methods for Managing the Classification and Solution of Data Imbalance Problem. *Journal of Computer Science,16*(11), 1546-1557.

Hickman, L., Thapa, S., Tay, L., Cao, M. & Srinivasan, P. (2020). Text preprocessing for text mining in organizational research: Review and recommendations. *Organizational Research Methods*, *25*(1), 114-146. https://doi.org/10.1177/1094428120971683

Hirsch, T., & Hofer, B. (2022). Using textual bug reports to predict the fault category of software bugs. *Array, 15,* 100189. https://doi.org/10.1016/j.array.2022.100189

Kang, S., Yoon, J., Askarbekkyzy, N., & Yoo, S. (2024). Evaluating Diverse Large Language Models for Automatic and General Bug Reproduction. *IEEE Transactions on Software Engineering, 50,* 2677-2694. https://doi: 10.1109/TSE.2024.3450837

Köksal, Ö. & Tekinerdogan, B. (2022). Automated Classification of Unstructured Bilingual Software Bug Reports: An Industrial Case Study Research. *Appl. Sci.*, *12*(1), 338. https://doi.org/10.3390/app12010338.

Iqbal, S., Naseem, R., Jan, S., Alshmrany, S., Yasar, M., & Ali, A. (2018). Determining Bug Prioritization Using Feature Reduction and Clustering With Classification. *IEEE Access, 8*, 215661–215678.

Juba, B., & Le, H. S. (2019). Precision-Recall versus Accuracy and the Role of Large Data Sets. *Proceedings of the AAAI Conference on Artificial Intelligence, 33*(01), 4039-4048. https://doi.org/10.1609/aaai.v33i01.33014039

Kukkar, A., & Mohana, R.M. (2018). A Supervised Bug Report Classification with Incorporate and Textual Field Knowledge. *Procedia Computer Science, 132,* 352-361. https://doi.org/10.1016/j.procs.2018.05.194

Li, R., Liu, M., Xu, D., Gao, J., Wu, F., & Zhu, L. (2022). A Review of Machine Learning Algorithms for Text Classification. In Proceedings of Lu, W., Zhang, Y., Wen, W., Yan, H., Li, C. (eds) Cyber Security. CNCERT 2021. *Communications in Computer and Information Science,* vol 1506. Springer, Singapore. https://doi.org/10.1007/978-981-16-9229-1_14

Lopes, F., Agnelo, J., Teixeira, C.A., Laranjeiro, N., & Bernardino, J. (2020). Automating orthogonal defect classification using machine learning algorithms. *Future Generation Computer Systems, 102,* 932-947. https://doi.org/10.1016/j.future.2019.09.009

Meng, F., Wang, X., Wang, J., Wang, P. (2022). Automatic Classification of Bug Reports Based on Multiple Text Information and Reports' Intention. In: Aït-Ameur, Y., Crăciun, F. (eds) Theoretical Aspects of Software Engineering. TASE 2022. Lecture Notes in Computer Science, vol 13299. Springer, Cham, 131- 147. https://doi.org/10.1007/978-3-031-10363-6_9

Paul, A., Mukherjee, D.P., Das, P., Gangopadhyay, A., Chintha, A.R., & Kundu, S. (2018). Improved Random Forest for Classification. *IEEE Transactions on Image Processing, 27*, 4012-4024. https://doi.org/10.1109/TIP.2018.2834830

Ramay, W.Y., Umer, Q., Yin, X., Zhu, C., & Illahi, I. (2019). Deep Neural Network-Based Severity Prediction of Bug Reports. *IEEE Access, 7,* 46846-46857. https://doi.org/ 10.1109/ACCESS.2019.2909746

Starbuck, C. (2023). Logistic Regression. In: The Fundamentals of People Analytics. Springer, Cham, pp. 223-238. https://doi.org/10.1007/978-3-031-28674-2_12

Soltani, M., Hermans, F.F., & Bäck, T. (2020). The significance of bug report elements. *Empirical Software Engineering, 25,* 5255 - 5294. https://doi.org/10.1007/s10664-020-09882-z

Steidl, G. (2015). Supervised Learning by Support Vector Machines. In: *Handbook of Mathematical Methods in Imaging,* Springer, New York, NY. https://doi.org/10.1007/978-3-642-27795-5_22-5

Subramani, P., Thiyaneswaran, B., Sujatha, M., Nalini, C., & Rajkumar, S. (2022). Grid Search for Predicting Coronary Heart Disease by Tuning Hyper-Parameters. *Comput. Syst. Sci. Eng., 43,* 737-749. https://doi.org/10.32604/csse.2022.022739

Tabassum, N., Namoun, A., Alyas, T., Tufail, A., Taqi, M., & Kim, K. (2023). Classification of Bugs in

Cloud Computing Applications Using Machine Learning Techniques. *Applied Sciences*, *13*(5), 2880. https://doi.org/10.3390/app13052880

Vito, G.D., Starace, L.L.L., Martino, S.D., Ferrucci, F., & Palomba, F. (2024). Large Language Models in Software Engineering: A Focus on Report Issue Classification and User Acceptance Test Generation. *Ital-IA 2024: 4th National Conference on Artificial Intelligence, organized by CINI, May 29-30, 2024, Naples, Italy*.

How to cite this Article:

Sara Khan and Saurabh Pal (2024). User Interface Bug Classification Model Using ML and NLP Techniques: A Comparative Performance Analysis of ML Models. *International Journal of Experimental Research and Review*, *45*, 56-69.

DOI : https://doi.org/10.52756/ijerr.2024.v45spl.005