IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 16 May 2023, accepted 14 June 2023, date of publication 21 June 2023, date of current version 28 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3288156

RESEARCH ARTICLE

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

SHATHA ABED ALSAEDI[1,2], AMIN YOUSEF NOAMAN[1], AHMED A. A. GAD-ELRAB[1], AND FATHY ELBOURAEY EASSA[1]

[1]Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Saudi Arabia
[2]Department of Computer Science, College of Computer Science and Engineering, Taibah University, Yanbu 46421, Saudi Arabia

Corresponding author: Shatha Abed Alsaedi (saalsaedi@stu.kau.edu.sa)

This work was supported by the Deanship of Scientific Research (DSR), King Abdulaziz University, Jeddah, under Grant KEP-PHD-102-611-1443.

ABSTRACT In software development systems, the maintenance process of software systems attracted the attention of researchers due to its importance in fixing the defects discovered in the software testing by using bug reports (BRs) which include detailed information like description, status, reporter, assignee, priority, and severity of the bug and other information. The main problem in this process is how to analyze these BRs to discover all defects in the system, which is a tedious and time-consuming task if done manually because the number of BRs increases dramatically. Thus, the automated solution is the best. Most of the current research focuses on automating this process from different aspects, such as detecting the severity or priority of the bug. However, they did not consider the nature of the bug, which is a multi-class classification problem. This paper solves this problem by proposing a new prediction model to analyze BRs and predict the nature of the bug. The proposed model constructs an ensemble machine learning algorithm using natural language processing (NLP) and machine learning techniques. We simulate the proposed model by using a publicly available dataset for two online software bug repositories (Mozilla and Eclipse), which includes six classes: Program Anomaly, GUI, Network or Security, Configuration, Performance, and Test-Code. The simulation results show that the proposed model can achieve better accuracy than most existing models, namely, 90.42% without text augmentation and 96.72% with text augmentation.

INDEX TERMS Software maintenance, nature classification, ensemble machine learning algorithm, natural language processing, bug reports, machine learning.

I. INTRODUCTION

In software engineering, testing is the evaluation process that is performed to indicate whether a specific system meets the requirements and concerns finding bugs or failures in meeting these requirements defined by the stakeholders [1]. As a result of this process, the maintenance phase fixes defects discovered after the termination of the testing phase. In addition, as the complexity and size of the software increase, software producers tend to release their software with defects [2], and software projects have a higher probability of having bugs. Therefore, users report these discovered defects and bugs [2]. A bug is a fault, an error, a failure, or a flaw in the software, which makes it behave incorrectly or generate wrong outputs [3]. The feedback of the reporter is sent to the bug tracking system (BTS) in the form of a bug report. Figure 1 illustrates an example of a bug report in the Eclipse repository.[1] A bug report contains information related to the discovered bug, such as bug ID, bug status (closed or opened), bug description, component affected by the bug, information about the software, how to reproduce the bug, bug reporter, and the developer who fix this bug [4].

A bug report can be thought as the medium that transfers and delivers the bug to the developers [5]. The process used by

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Liu.

[1]https://bugs.eclipse.org/bugs/show_bug.cgi?id=220151

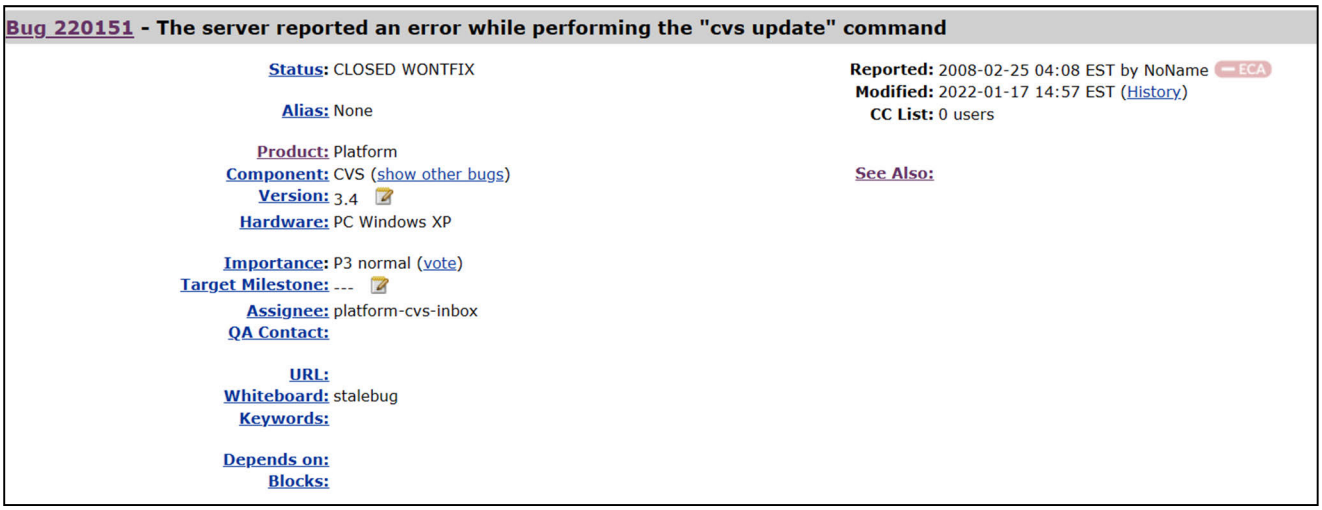


FIGURE 1. Eclipse bug report 220151.

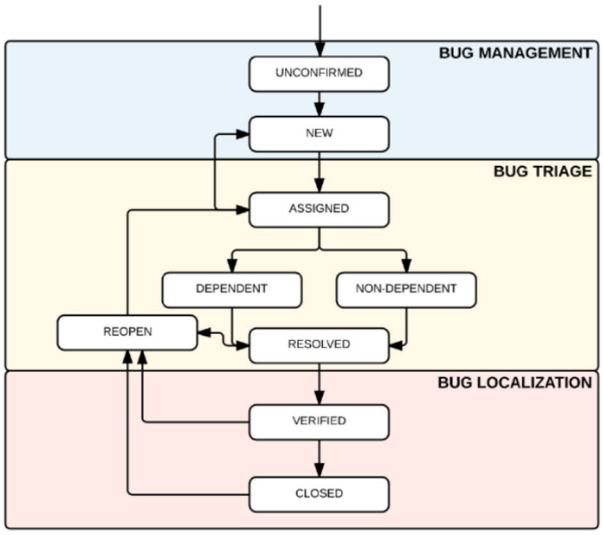


FIGURE 2. Life cycle of the bug report [7].

the developer after the assignment of the bug report to resolve it is the bug management process [6]. This process starts when this report is submitted by users to the bug management system when they face an error in a released software product.

Then, this bug report is assigned to developers who work to find the location of the bug. The bug is fixed by the developer who finds the cause of the bug and its location before other developers. After the bug resolution, the tester checks the bug scenario, and if it does not reoccur, updates the bug report status to Verified. Finally, the reporter receives a notification [6].

A software bug has its own life cycle made of different phases in its life. Therefore, Figure 2 illustrates the bug report life cycle. This figure shows that the life cycle has three phases, bug management, bug triage, and bug localization. The bug management phase includes all activities which start when users report the bug until the assignment of the developer. The next phase is bug triage when submitted reports are prioritized and assigned to a suitable developer for repair. Finally, the bug localization phase changes the bug status from resolved to verified to closed [7].

In this life cycle, the main challenge is the dramatically increasing number of bug reports, which are tedious, cumbersome, and time-consuming to manage manually [8]. Researchers address this issue by analyzing and classifying these reports according to three major categories, each with its sub-categories [9], and extracting useful information to accelerate and facilitate the maintenance phase. These categories of bug reports are classified by nature, priority, and severity [9]. Most studies classify bug reports by severity or priority.

The literature review in this research shows that various classification algorithms classify bug reports from different aspects, but there is a lack of nature-based bug classification models with high accuracy.

Therefore, this paper intends to bridge this gap by introducing an ensemble machine learning algorithm for nature-based bug prediction from bug reports.

This research aims to provide a solution to maintaining software systems and contribute to this context by applying natural language processing (NLP), machine learning (ML), and text mining techniques to predict the bug types automatically. Once the model identifies the bug type, it accelerates the maintenance phase with bug localization techniques rather than doing this time-consuming task manually. Figure 3 illustrates an overview of the bug prediction process from a bug report.

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. Additionally, we suggest ensemble machine learning classifiers, using both hard and soft voting, to improve the

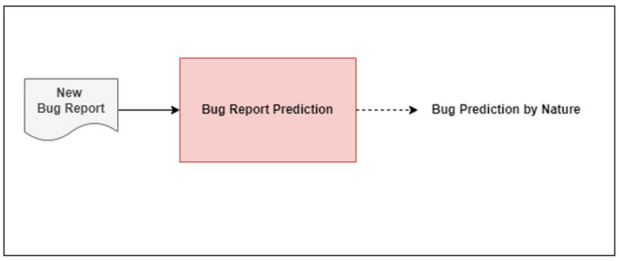


FIGURE 3. Nature-based prediction.

performance of the proposed model. Moreover, we use a text augmentation technique to boost the accuracy.

The main contributions of this paper are as follows:

- It presents an evaluation of some base machine learning algorithms for an automatic nature-based bug report classification.
- It proposes an automated ensemble machine learning-based approach for nature-based bug prediction from bug reports.
- It introduces a text augmentation technique in nature-based bug prediction from bug reports in the proposed model. To the best of our knowledge, it is the first ensemble machine learning approach to predict the nature of bugs in bug reports using a text augmentation technique.
- The evaluation results of the proposed approach on a benchmark dataset suggest that the proposed ensemble machine learning approach is accurate in nature-based bug prediction from bug reports.

II. RELATED WORK

Several studies in the literature aim to apply natural language processing, information retrieval, and artificial intelligence in the context of bug report analysis. Each has a vital role in software maintenance phase enhancements. Here is an illustration of some recent studies based on bug report classification and bug report assignments.

A. BUG REPORTS CLASSIFICATION

In [3], the author proposed a tool to analyze a new bug report and construct configuration options. This tool detects whether this report is configurable or not. If it is related to a configuration bug, it ranks the top possible configurations related to this bug. However, the configuration identification phase has its limitation in ranked configurations with varying words as the tool chooses configuration with longer words even though the configuration with shorter words is the correct one.

Kukkar and Mohana [10] combine text mining, natural language processing, and machine learning techniques to classify bug and non-bug reports to solve the problem of misclassification of bug reports, which negatively affects the overall performance of the prediction process. This model uses bigram and TF-IDF in feature selection. However, the performance of the KNN algorithm in this model changes when the data-set changes.

In addition, as the number of software increases frequently, the size of bug-tracking system repositories increases significantly [8]. Therefore, it is urgent to fix the bugs which have higher severity first. Since the number of bug reports is very high, it is difficult to analyze them manually. Therefore, much research has focused on automating bug report severity detection. In [8], the authors proposed an algorithm based on deep learning and random forest with boosting to assign a severity level for each bug report. Their method has higher accuracy than other methods that address the same problem because the proposed method uses a convolutional neural network (CNN) for the feature extraction, then uses random forest with boosting for severity classification. The average achieved accuracy was 96.34%.

Some researchers have different classifications for bug reports. In [11], researchers use machine learning algorithms to classify new bug reports as either corrective (defect fixing) reports or perfective (major maintenance) reports by using support vector machines (SVM), naive Bayes (NB), and random trees (RT). The results show that SVM achieved the highest accuracy, 93.1%.

To support the software engineering process, researchers have introduced a well-known schema called Orthogonal Defect Classification (ODC) [12]. ODC has eight orthogonal attributes to characterize software defects and several analytical methods for test process analysis and software development [51]. Extracting valuable information from defects can be done by ODC, which provides insights and helps diagnosis in software engineering processes [13]. The authors in [13] use machine learning algorithms to classify bug reports according to ODC. Their study uses 4096 ODC annotated bug reports. However, they conclude that there is difficulty in automating the ODC attributes using only bug reports.

Hirsch and Hofer [14] classified each new bug according to a bug report using three classes, concurrency, memory, and semantic bugs. They used 369 bug reports, and the highest mean precision and recall were 0.74 and 0.72, respectively.

Furthermore, bug prioritization is a crucial feature of bug reports because users submit many bug reports for the developer to address [15]. Manually assigning a priority level for each bug report is time-consuming and needs expertise, time, and resources [16] because it is more likely to be assigned incorrectly, which affects the maintenance phase. Therefore, some studies focus on finding the priority of bug reports automatically using machine learning algorithms [15]. In [16], Bani-Salameh et al. proposed a method by using NLP and deep learning-based algorithms in bug report triage. The proposed model predicts and assigns a priority level in binary classification (high or low) to each bug report using a five-layer RNN-LSTM neural network to classify each bug report as a high or a low priority. The proposed model was evaluated by applying it to a dataset with more than 2000 bug reports from the JIRA dataset. However, the authors express concern about whether the performance of this model will be the same when using other datasets from different resources.

TABLE 1. Summary of previous studies on bug reports processing.

Reference	Idea	Dataset	Dataset size	Algorithms / Software package	Evaluation
[3]	Predicts each bug report as (Configuration) or (Non-configuration) bug reports and, if it is configurable, outputs the associated configuration names.	Mozilla, MySQL, and Apache.	900	Weka, NLTK, and Sklearn software packages	Accuracies of relating configuration bug reports with configurations are: Mozilla 0.92 Apache 0.88 MySQL 0.74
[10]	Predicts each bug report as (Bug) or (non-bug) report	Mozilla, Eclipse, JBoss, Firefox, and OpenFOAM	3200	Bigram + TF-IDF + Info gain + K-NN	The values of recall, precision, and F-measure are changed depending on the bug report dataset.
[8]	Predicts the severity of the bug reports	Mozilla, Eclipse, JBoss, OpenFOAM, and Firefox	3220	N-gram, CNN, and Random Forest with Boosting	The average accuracy is 96.34%. The average F-measure is 96.43%.
[11]	Predicts each bug report to be either a Corrective (defect fixing) report or a Perfective (major maintenance) report.	AspectJ, Tomcat, and SWT	5800	NB- SVM-RT	Achieve higher accuracy using SVM, which is about 93.1%
[13]	Classifies bug reports according to ODC.	MongoDB, Cassandra, and HBase	4096	NB, SVM, KNN, RNN, nearest centroid, and RF	The size of the dataset affects the accuracy of the classifiers.
[14]	Predicts bug type in the main category only (Concurrency, Memory, and Semantic bugs)	GitHub projects	369	Multinomial Naive Bayes (MNB), Linear Support Vector with Stochastic Gradient Descent learning (SGDC), Linear Support Vector (LSVC), Random Forrest (RFC), and Logistic Regression (LRC)	The highest score of mean precision is 0.74, and recall is 0.72.
[16]	Predicts and assigns a priority level in binary classification (high or low)	JIRA	More than 2000	RNN-LSTM	Accuracy is 0.908, AUC is 0.95, and F-measure is 0.892
[17]	Classifies each bug report into a class from (assignment/initialization, external interface, internal interface, and other).	Proprietary Bug Dataset	504	NB, SVM, KNN, LR, DT, and RF	Accuracy is 73.70%
[19]	A model for bug Assignment recommendation	Collected data are from one development project at Company Automation and four major development projects at Company Telecom.	35,266	Stacked Generalization (SG) which is an ensemble learning technique (NB, SVM, KNN, DT and Bayes Net)	Accuracies are from 50% to 89%
[18]	A model for adaptive ranking which ranks the top developers	Birt, Eclipse UI, JDT, and SWT	22,416	Integrate [20] and [21] approaches	In Birt and Eclipse UI, the model correctly ranked developer within the top-5 developers by more than 80%.
[22]	Web-based tool for Bug assignment recommendation	Plasmashell, LibreOffice, and Firefox.	3034	SVM, Naive Bayes, C4.5, and Rules	The achieved accuracies are from 50% to 95%, from 20% to 80%, and from 10% to 70% for top-1, top-3, and top-5 recommendations, respectively.
[23]	A model for Bug assignment recommendation	NetBeans, Freedesktop, and Firefox	17117	Gain ratio and machine learning algorithm	The best F-Score achieved is 67%.

Moreover, the authors in [17] focused on analyzing bilingual software bug reports and applied their proposed algorithm to an industrial case study, which is a commercial software system, and it contains bug reports in English

and Turkish languages. 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.

B. BUG REPORTS ASSIGNMENTS

Numerous endeavors in the research sector have enhanced the quality of maintenance and decreased bug-fixing time. As manually assigning a bug fixer is time-consuming due to the increasing number of submitted bug reports, researchers recommend the developer, who is more suitable for fixing the bug using multiple approaches. This process is called bug assignment [18], and each approach has its significance and contribution.

In [19], Jonsson et al. proposed a new model using an ensemble machine learning classifier, namely Stacked Generalization (SG), in the bug assignment process. The proposed model uses at least 2000 bug reports in the training phase and achieves from 50% to 89% accuracy in prediction.

Furthermore, Alkhazi et al. [18] used the commits of the developers to enhance their profiling and designed an approach using the integration of approaches from [20] and [21] for adaptive ranking, which ranks the top developers who are better for the bug fixing in the new bug report. They evaluated their approach using about 22,000 bug reports.

Additionally, Devaiya [22] proposed a bug assignment tool, namely Creation Assistant for Supporting Triage Recommenders (CASTR), to recommend the most suitable developers for bug fixing. In other words, the author gives the ability in configuring project-specific parameters when establishing the recommender by using machine learning algorithms in tool designing. These algorithms are SVM, Naive Bayes, C4.5, and Rules. In the proposed tool, the achieved accuracy is from 50% to 95%, from 20% to 80%, and from 10% to 70% for top-1, top-3, and top-5 recommendations, respectively.

Other researchers focus on using categorical features in bug reports and use them in the recommendation system. Alenezi et al. [23] used a classification algorithm in designing a bug assignment algorithm and they used categorical fields of the bug reports without a textual description. However, they achieve relatively low results. The best F-Score achieved is 67%, which leads them to conclude that using classification algorithms in bugs assignment is not the best approach. Table 1 summarizes all the previously mentioned studies.

III. THE PROPOSED NATURE-BASED ENSEMBLE MACHINE LEARNING BUG PREDICTION MODEL

This section describes the basic idea of our model and the suggested methodology to be applied to achieve the main objective, which is nature-based prediction of bug report.

A. BASIC IDEA

To improve the process of nature-based classification of bug reports, this paper proposes a new prediction model called

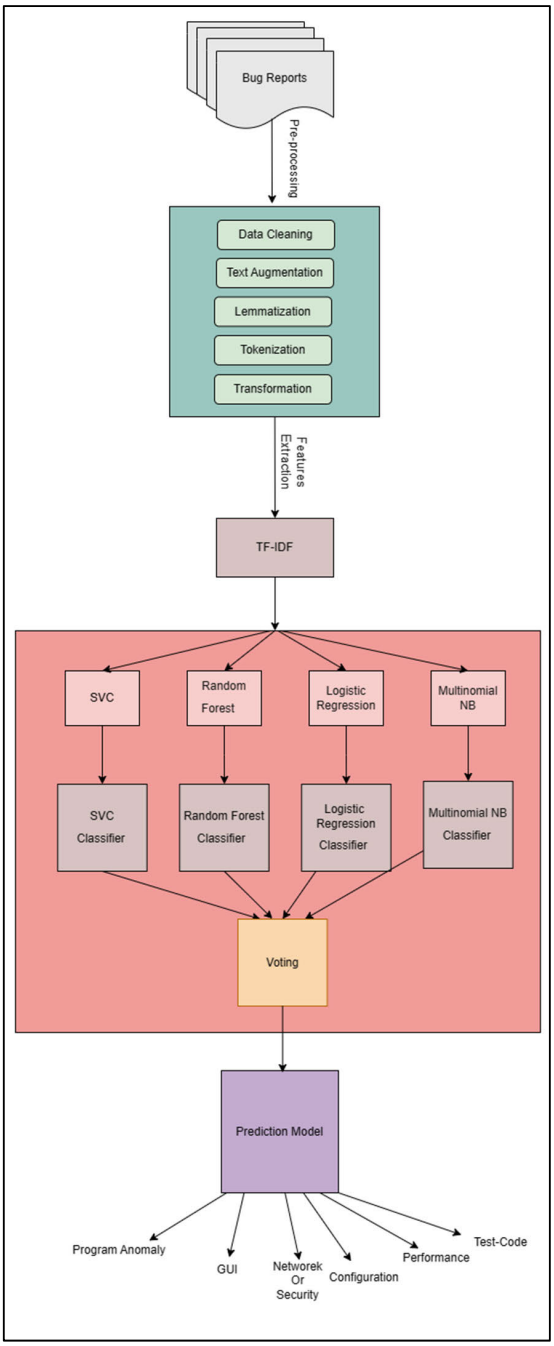


FIGURE 4. Overall architecture for proposed model.

Nature-Based Ensemble Machine Learning Bug Prediction Model. The basic idea of the proposed model depends on three issues: 1) using a feature extraction approach to extract the most critical features that will affect the bug report prediction, 2) training several base machine learning algorithms, and 3) using a voting ensemble learning classifier by combining their prediction to improve the prediction accuracy.

B. METHODOLOGY

This section satisfies the basic idea of the proposed model with a methodology of four phases: 1) Data pre-processing,

2) Vectorization and feature extraction, 3) Training set of base machine learning classifiers, and 4) Building a voting ensemble machine learning classifier for nature-based prediction. The overall model is illustrated in Figure 4. The following subsections present a clear illustration of each step.

1) DATA PRE-PROCESSING

The pre-processing mechanism cleans and prepares the raw data of the bug report for the machine learning algorithm processing to extract valuable and interesting data and knowledge from unstructured data [28]. Text pre-processing eliminates unnecessary words and tokens and reduces the feature set size to enhance the learning process [29]. In the model, the following steps has been done in this phase:

- The bug reports have been read in comma-separated values (CSV) format.
- The classification attribute has been selected which is the 'Category' field in the CSV file.
- Textual attribute, which is the 'Summary' field, has been selected as a textual feature.
- Text cleaning: removes unnecessary text, such as punctuation, numbers, extra space, and emojis [30].
- Lowercase Transformation: It converts the input text to lower case [30]. For example, 'Fix', 'FIX' and 'fIX' all become 'fix' This phase is done because machine learning algorithms are case-sensitive [31]. The same word in different cases will be treated differently by statistical models [32] although they have the same meaning.
- Tokenization: This mechanism separates or splits the words and sentences of the text into their smallest fragments, which are called tokens [33]. These tokens cannot divide further into smaller tokens [33].
- Stop words removal: This phase removes the stop words from our text because they do not carry useful meaning to the natural language processing [34]. These stop words include articles, prepositions, conjunctions, pronouns, adjectives, and adverbs.
- Lemmatization: In this phase, every word has been reduced to its morphological form (lemma) [35]. For instance, 'write', 'wrote', and 'writing' can be replaced with 'write'. Table 2 shows some text in the used bug reports dataset and the effects of each pre-processing activity on it.
- Text Augmentation: we use text augmentation to achieve higher accuracy. Text augmentation is a technique that artificially increases the training data size by producing different versions of real datasets without actually collecting the data [46]. Text augmentation can be applied on various levels, such as character, word, phrase, and document levels [46]. There are several methods for text augmentation, such as synonym replacement, replace words with similar word embeddings, lexical-based replacement, back translation, and generative models [47]. Our proposed model replaces words with similar word embeddings. In this method, trained word embedding, such as fastText, Word2Vec, and GloVe, can help identify the closest word vector to replace the original sentence from latent space [47]. In more details, contextual word embeddings augmenter has been used.

TABLE 2. Pre-processing activities and effects on a sample of text in the dataset.

Phase	Text
Original Text	`Always show Welcome at start up label is not clickable`
Text Cleaning	`Always show Welcome at start up label is not clickable`
Lower Case	`always show welcome at start up label is not clickable`
Tokenization	`'always' 'show' 'welcome' 'at' 'start' 'up' 'label' 'is' 'not' 'clickable'`
Stop word removal	`'always' 'show' 'welcome' 'start' 'label' 'clickable'`
Lemmatization	`'always' 'show' 'welcome' 'start' 'label' 'clickable'`

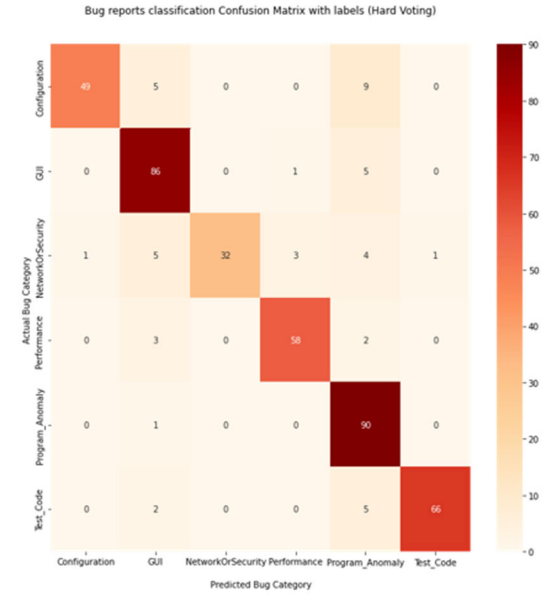
TABLE 3. Number of bug reports in each category in the used dataset.

Nature category	Number of bug reports
Program Anomaly	498
Performance	316
Configuration	281
Network or Security	242
GUI	465
Test Code	336
Total Number of Bug Reports	2138

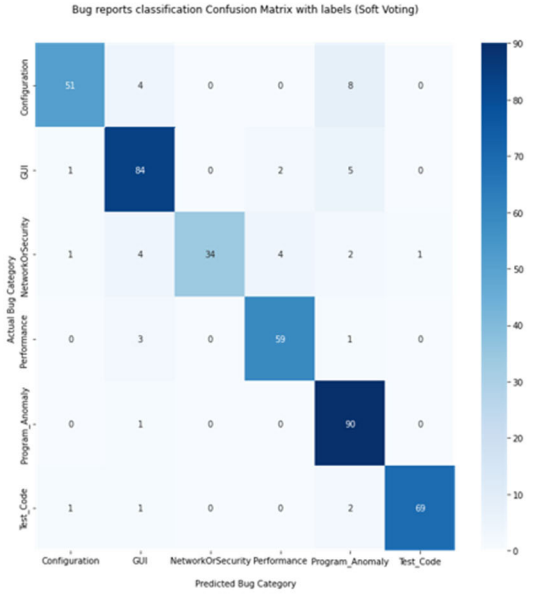
2) VECTORIZATION AND FEATURE EXTRACTION

Feature extraction transforms the content of bug reports into a vector of feature (word) counts [36]. This process aids in expanding the network by converting the content of bug reports into multiple sets of n-gram data [36]. Therefore, it provides a meaningful way to express the features [36]. The n-gram technique has been implemented to calculate the frequency of the feature order and capture the semantic relationship [36].

Words which are resulted after pre-processing are called features [8]. The feature extraction phase represents the contents of the bug report as a vector of words (features) counts by transforming the contents of bug reports into several sets of n-gram data that help expand the network [8]. Moreover, this phase transforms the word frequency to give a score or identification [50]. For each token, the Term Frequency–Inverse Document Frequency (TF-IDF) is used with a unigram score. Equation (1) defines TF-IDF. Where in a document set, t is the



(a) Hard voting ensemble ML algorithm



(b) Soft voting ensemble ML algorithm

FIGURE 5. Confusion matrices for two voting machine learning classifiers (without text augmentation).

term of document d, n is the total number of documents in the document set, and the document frequency of t is df(t) [36].

$$\text{tf} - \text{idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t) \quad (1)$$

$$\text{idf}(t) = \log[\frac{n}{df(t)}] + 1 \quad (2)$$

3) TRAINING SET OF BASE MACHINE LEARNING CLASSIFIERS

Before training the dataset using machine learning classifiers, dataset needs to be splitting into training and testing sets. We divide the dataset into 8:2 ratio for training and testing.

TABLE 4. An example of bug report summary which is related to each category.

Nature category	Summary of bug report
Program Anomaly	AST parser returns error
Performance	Website runs with 100% CPU in Firefox without showing a slow script warning
Configuration	When RCP is upgraded from Eclipse3.4.2 to Eclipse4.6.3, one UI program has UI freeze issue.
Network or Security	[e10s] IPv6 sockets can fail to bind or connect
GUI	Wrong button name in "Workspace Unavailable" Dialog
Test Code	The junit test FindReplaceDialogTest.testFocusNotChangedWhenEnterPressed fails on Mac

TABLE 5. Results comparison without text augmentation.

Reference	Algorithm	Accuracy
Benchmark work [24]	Random Forest	88.78%
	Naïve Bayes	67.05%
	Decision Tree	83.87%
	Logistic Regression	85.51%
Algorithms in this paper	Random Forest	88.55%
	Multinomial Naïve Bayes	83.64%
	Support Vector Classification	88.78%
	Logistic Regression	88.08%
	Proposed ensemble model (Hard voting)	89.01%
	Proposed ensemble model (Soft voting)	**90.42%**

Our model uses four machine learning (ML) algorithms to train the training dataset, Random Forest (RF), Logistic Regression (LR), Multinomial Naïve Bayes, and Support Vector Classifier (SVC) algorithms.

a: RANDOM FOREST (RF) CLASSIFIER

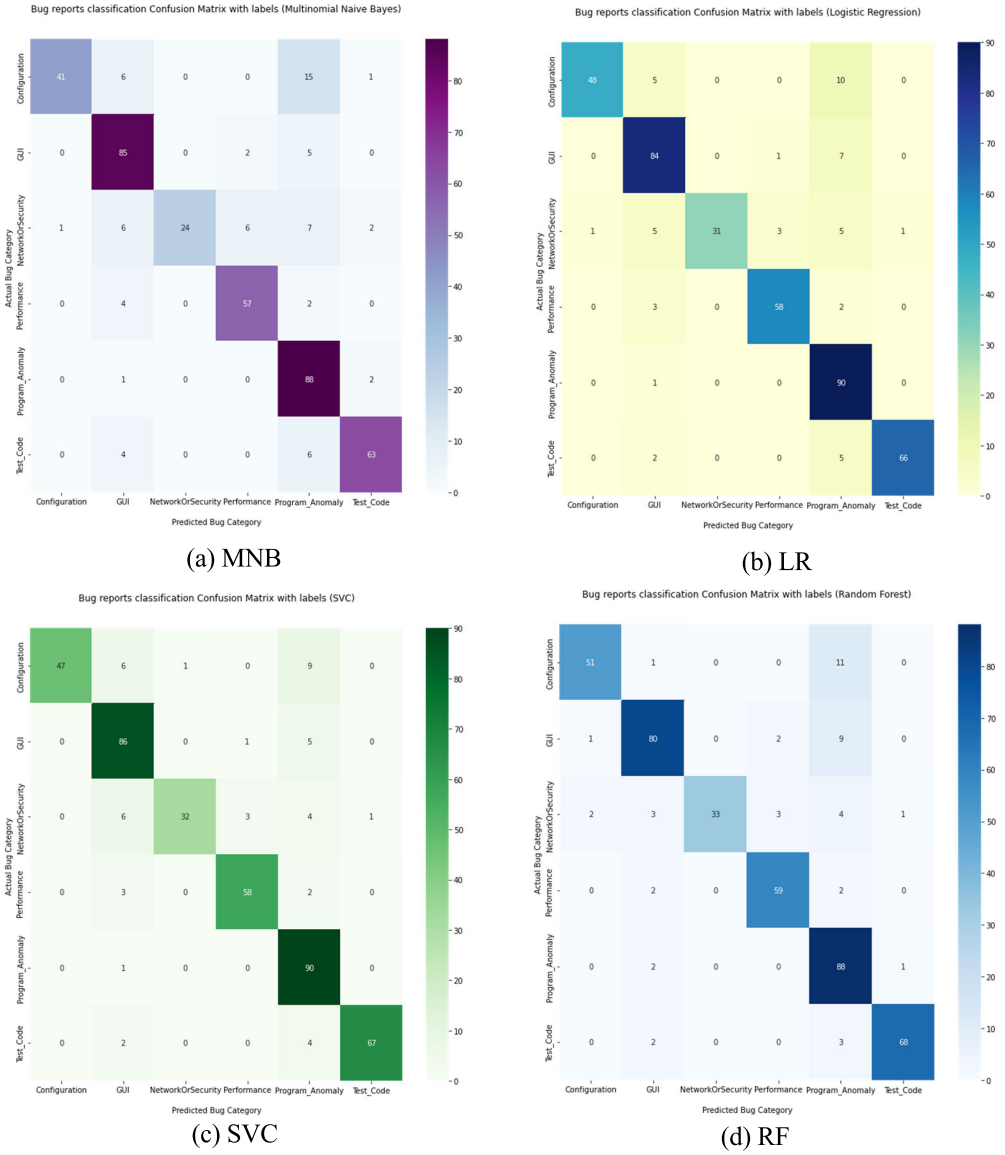
A random forest (RF)[2] is a machine learning algorithm. This algorithm is a meta-estimator that fits several decision tree classifiers on many datasets subsamples [37]. Additionally, it uses averaging to control over-fitting and improve predictive accuracy [37].

b: LOGISTIC REGRESSION (LR) CLASSIFIER

Logistic regression (LR)[3] algorithm is a machine learning statistical algorithm. It builds a logistic model known as a

[2]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[3]https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html



(a) MNB (b) LR (c) SVC (d) RF

FIGURE 6. **Confusion matrices for base machine learning classifiers (without text augmentation).**

logit model [38]. One advantage of this model is its ability to be used for both class probability estimation and classification as it relates to the distribution of logistic data [38]. The LR model applies a nonlinear sigmoidal function to a linear combination of features [38]. The base version of LR is applied to binary classification problems, but it can be extended into multi-classes classification problems (known as multinomial logistic regression) [38].

c: MULTINOMIAL NAÏVE BAYES (MNB) CLASSIFIER

Naïve Bayes (NB)[4] classifier is a machine learning algorithm that applies the Bayes theorem [39]. This probabilistic algorithm can be used in classification problems. Only a minimal training dataset is necessary to estimate the parameters for classification since this classifier assumes that all variables are independent [39]. There are many types of NB models, such as Gaussian Naïve Bayes, Multinomial Naïve Bayes, and Bernoulli Naïve Bayes [40]. The proposed model utilizes the Multinomial Naïve Bayes classifier, usually used in document classification problems. The classifier uses the feature of the frequency of the words that appear in the document [40].

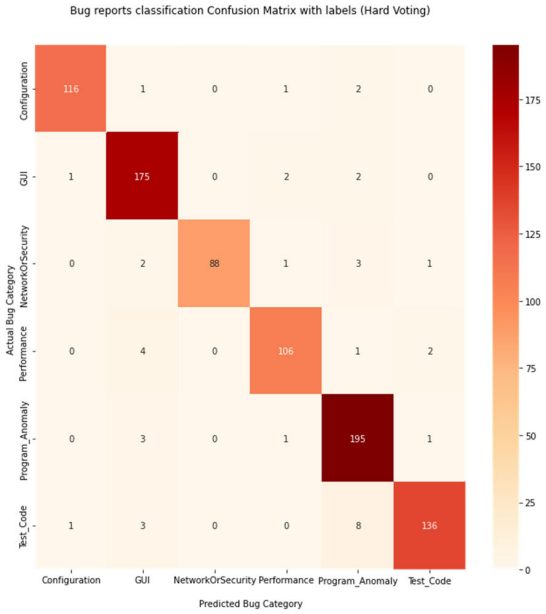
d: SUPPORT VECTOR CLASSIFIER (SVC) CLASSIFIER

The support vector machine (SVM) model is a powerful and flexible supervised machine learning method for outliers detection, classification, and regression problems [41]. It is efficient in high-dimensional spaces, so it is helpful

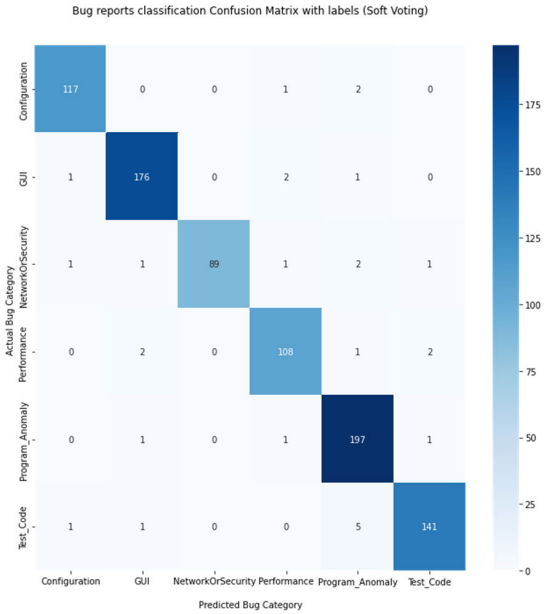
[4]https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

TABLE 6. Analysis of results of algorithms (without text augmentation).

Algorithm in this paper	Class	Precision	Recall	F1-Measure
Random Forest	Configuration GUI Network or Security Performance Program Anomaly Test Code	94% 89% 100% 92% 75% 97%	81% 87% 72% 94% 97% 93%	87% 88% 84% 93% 85% 95%
Multinomial Naïve Bayes	Configuration GUI Network or Security Performance Program Anomaly Test Code	98% 80% 100% 88% 72% 93%	65% 92% 52% 90% 97% 86%	78% 86% 69% 89% 82% 89%
Support Vector Classification	Configuration GUI Network or Security Performance Program Anomaly Test Code	100% 83% 97% 94% 79% 99%	75% 93% 70% 92% 99% 92%	85% 88% 81% 93% 88% 95%
Logistic Regression	Configuration GUI Network or Security Performance Program Anomaly Test Code	98% 84% 100% 94% 76% 99%	76% 91% 67% 92% 99% 90%	86% 87% 81% 93% 86% 94%
Proposed ensemble model (Hard voting)	Configuration GUI Network or Security Performance Program Anomaly Test Code	98% 84% 100% 94% 78% 99%	78% 93% 70% 92% 99% 90%	87% 89% 82% 93% 87% 94%
Proposed ensemble model (Soft voting)	Configuration GUI Network or Security Performance Program Anomaly Test Code	94% 88% 100% 92% 82% 97%	81% 91% 76% 94% 98% 95%	87% 89% 86% 93% 89% 96%



(a) Hard voting ensemble ML algorithm



(b) Soft voting ensemble ML algorithm

FIGURE 7. Confusion matrices ensemble voting machine learning classifiers (with text augmentation).

for classification problems [41]. SVM is useful for binary classification, and as the proposed model includes multiclass classification, it uses the Support Vector Classifier (SVC)[5] algorithm because it can perform multiclass classification.

4) BUILDING VOTING ENSEMBLE MACHINE LEARNING CLASSIFIER

An ensemble learning system is a hybrid learning system [43]. It trains several base learning algorithms as ensemble members and combines their predictions into a single output. This result performs better on average than other ensemble members [42]. There are many types of ensemble learning, such as bagging, boosting, and voting [44]. In voting ensemble learning, voting (known as majority voting) can be either hard or soft voting [45]. In classification problems with hard voting, the votes for crisp class labels from other models are summed, and then the class with the most votes is predicted [45]. On the other hand, in soft voting, the predicted probabilities for class labels are summed, and the class label with the highest probability sum is predicted [45]. In the experiment of the proposed model, we implemented hard voting and soft voting,[6] but the latter achieved the highest accuracy, as illustrated in the results section.

IV. EVALUATION AND RESULTS

This section provides a detailed description of the implementation and evaluation process of the proposed model.

A. SIMULATION SETTINGS

The experiments in this research are implemented in Python. The dataset is in comma-separated values (CSV) format. Pandas package is used for dataset handling. Natural Language Toolkit (NLTK) is used for the dataset pre-processing phase, such as lemmatization and stop word removal. Additionally, nlpaug library is used for text augmentation. Scikit-learn is used because it contains many packages, including feature representation, classification using base ML classifier and ensemble ML algorithm, and evaluation metrics.

B. DATASET DESCRIPTION

The dataset used in this research from two online bug repositories, Mozilla[7] and Eclipse,[8] which are in the Bugzilla bug tracking system and labeled by [24]. These authentic, open-source bug repositories contain many bug reports [24]. The category of each bug report is not mentioned in these repositories [24]. Therefore, authors in [24] randomly selected around 2000 bug reports and manually labeled them based on six categories (bug nature). These bug reports were submitted to the bug tracking system in the period between 2016 to 2019.

Each bug report consists of categorical and textual features. The categorical features include Bug ID, product, component, assignee, status, classification, priority, opened, and category attributes. The textual feature includes the summary attribute. The summary attribute is chosen as the textual feature because it contains a detailed description of the reported bug.

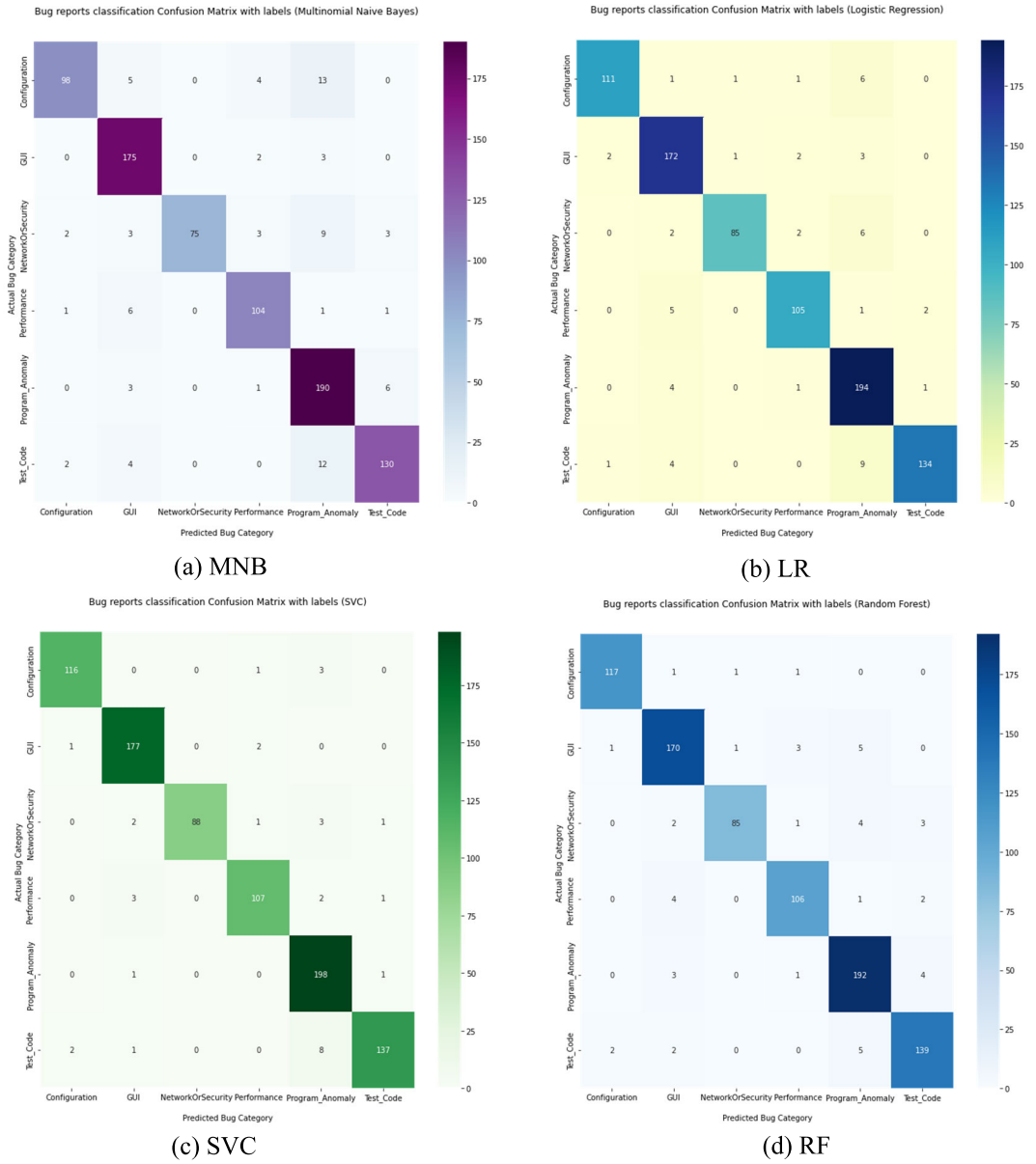
The classification of bug reports depends on the nature of the bug, therefore, this dataset was labeled using six categories, Program Anomaly, GUI, Network or Security,

[5]https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[6]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html

[7]https://bugzilla.mozilla.org/home

[8]https://bugs.eclipse.org/bugs/



(a) MNB (b) LR (c) SVC (d) RF

FIGURE 8. Confusion matrices for base machine learning classifiers (with text augmentation).

Configuration, Performance, and Test-Code. These categories are as follows:

1) Program Anomaly: This category involves bugs due to source code problems [24]. Exceptions, syntax errors, logical errors, and return value problems [25] are examples of program anomaly problems. One example of a bug report in this category is when the AST parser returns an error.

2) GUI: This category involves bugs that occur specifically during the design and handling of user interfaces [24]. An example of bugs from this category is when there is an error in naming the "Workspace Unavailable" dialog.

3) Network or Security: This category involves security issues or network problems [24]. An example of a bug related to a network problem is when there is a failure to bind or connect IPv6 sockets. An example of a bug related to a security problem is when a user wants to access the windowUtils property, and permission is denied [24].

4) Configuration: Bugs related to this category occur due to the integration of configuration files. An example of a bug from this category occurs due to a problem when updating an application, which includes missing of shared configuration area, after the application has been updated [24].

5) Performance: This category involves problems related to memory, which include infinite loops that lead to hanging up of memory, energy leaks, and extra memory usage [26]. One example of reported bug from this category is the issue in Firefox when the website runs 100% of its CPU without displaying a slow script warning.

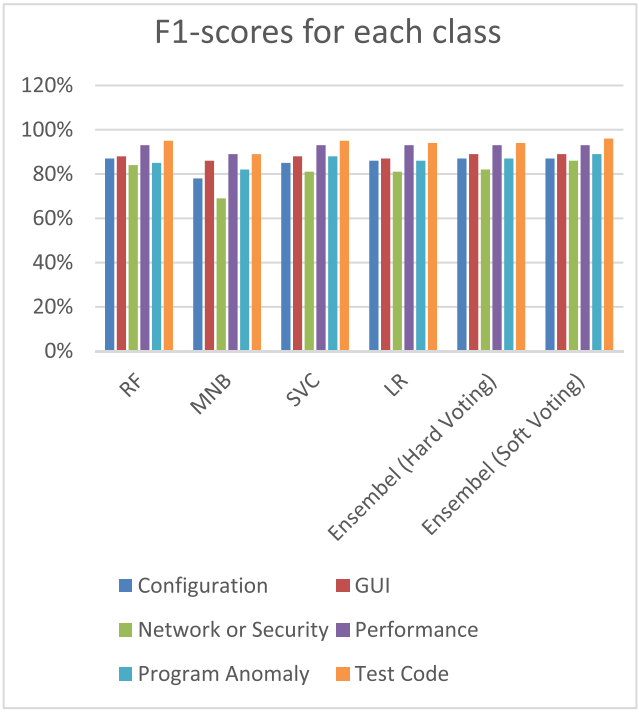


FIGURE 9. F1-scores for each class without text augmentation.

6) Test Code: Bugs in this category involve problems in the test code [24]. In the dataset, test code bugs occur due to i) intermittent tests, ii) test cases running, repairing, and updating, and iii) the failure of tests that happen when searching for de-localized bugs [27]. An example of a bug in this category is the issue that appears due to the Junit test failing on a Mac.

Tables 3 and 4 show the number of bug reports in each category in the used dataset and an example of a bug report summary related to each category, respectively.

C. EVALUATION METRICS

Generally, the accuracy and performance of classification algorithms are evaluated using different performance evaluation metrics, such as accuracy and recall [48]. Additionally, for evaluation the performance of machine learning classifiers, a confusion matrix can be used. It gives results about the actual and predicted classification achieved by the classifier model [48]. In this research, we used four well-known evaluation metrics which are accuracy, precision, recall, and F-measure. These evaluation metrics are obtained using a confusion matrix [48]. The four values, which are true positive (TP), true negative (TN), false positive (FP), and false negative (FN) are in the confusion matrix. The (TP) counts the actual positive values predicted by the classifier. The (TN) value means the actual negative values predicted by the classifier. (FP) are the values which are negative but were predicted as positive by the classifier, and (FN) are positive values that were predicted as negative by the classifier [48].

TABLE 7. Results comparison (with text augmentation on our work).

Reference	Algorithm	Accuracy
Benchmark work [24]	Random Forest	88.78%
	Naïve Bayes	67.05%
	Decision Tree	83.87%
	Logistic Regression	85.51%
Algorithms in this paper	Random Forest	94.50%
	Multinomial Naïve Bayes	90.18%
	Support Vector Classification	96.14%
	Logistic Regression	93.57%
	Proposed ensemble model (Hard voting)	95.32%
	Proposed ensemble model (Soft voting)	**96.72%**

1) ACCURACY

Accuracy is the percentage of correctly predicted classification to the total number of data [16]. This ratio is an important performance measure when using asymmetric datasets, which can exist when false positives and false negatives are the same value [49]. Accuracy can be calculated using the following equation:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}} \tag{3}$$

2) RECALL

Recall is the fraction of correctly predicted positive values to the same class's total observation [16]. It is calculated using the following equation:

$$\text{Recall} = \frac{\text{TP}}{\text{FN} + \text{TP}} \tag{4}$$

3) PRECISION

Precision is the ratio of correctly labeled positives to the total values which are predicted positive [16]. Precision can be

TABLE 8. Analysis of results of algorithms with text augmentation.

Algorithm in this paper	Class	Precision	Recall	F1-Measure
Random Forest	Configuration GUI Network or Security Performance Program Anomaly Test Code	97% 93% 98% 95% 93% 94%	97% 94% 89% 94% 96% 94%	97% 94% 93% 94% 94% 94%
Multinomial Naïve Bayes	Configuration GUI Network or Security Performance Program Anomaly Test Code	95% 89% 100% 91% 83% 93%	82% 97% 79% 92% 95% 88%	88% 93% 88% 92% 89% 90%
Support Vector Classification	Configuration GUI Network or Security Performance Program Anomaly Test Code	97% 96% 100% 96% 93% 98%	97% 98% 93% 95% 99% 93%	97% 97% 96% 96% 96% 95%
Logistic Regression	Configuration GUI Network or Security Performance Program Anomaly Test Code	97% 91% 98% 95% 89% 98%	93% 96% 89% 93% 97% 91%	95% 93% 93% 94% 93% 94%
Proposed ensemble model (Hard voting)	Configuration GUI Network or Security Performance Program Anomaly Test Code	98% 93% 100% 95% 92% 97%	97% 97% 93% 94% 97% 92%	97% 95% 96% 95% 95% 94%
Proposed ensemble model (Soft voting)	Configuration GUI Network or Security Performance Program Anomaly Test Code	97% 97% 100% 96% 95% 97%	97% 98% 94% 96% 98% 95%	97% 98% 97% 96% 97% 96%

calculated using the following formula:

$$\text{Precision} = \frac{\text{TP}}{\text{FP} + \text{TP}} \tag{5}$$

4) F1-MEASURE

It is the average recall and accuracy considering FP and FN [16]. This metric is more effective than accuracy, especially if the distribution is unbalanced [16]. It can be measured using the following equation.

$$\text{F1measure} = \frac{2 * \text{Precision} + \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6}$$

D. RESULTS AND ANALYSIS

This section shows the results of the proposed model and its evaluation using the dataset. We evaluated the performance of the proposed nature-based prediction model by two experiments: 1) without text augmentation, in which the used dataset has 2138 bug reports, and 2) with text augmentation, in which the used dataset has 4276 bug reports after applying text augmentation. In addition, four base machine learning algorithms were chosen in the experiments to be used on the dataset to predict the nature of bugs. These algorithms are RF, MNB, SVC, and LR. The rest of this section presents the results of two experiments.

1) WITHOUT TEXT AUGMENTATION

Table 5 presents the achieved accuracy of the proposed model and the four benchmark ML models. As illustrated in Table 5, before applying text augmentation, the highest accuracy is 90.42%, achieved by the proposed ensemble machine learning classifier using soft voting. While the proposed model reached 89.01% accuracy using hard voting, Figure 5 illustrates the confusion matrices for two voting machine learning classifiers. Other base ML classifiers worked moderately well and achieved an accuracy of 83.64%, 88.08%, 88.55%, and 88.78% for MNB, LR, RF, and SVC, respectively. In Figure 6, matrices a, b, c, and d illustrate confusion matrices for each base ML classifier.

Additionally, results in this paper have been evaluated using precision, recall, and F1-measure. Table 6 shows these results for each class in all base ML algorithms and the proposed voting ensemble algorithm without applying text augmentation.

2) WITH TEXT AUGMENTATION

Text augmentation has been applied to the dataset to increase the performance of the proposed model. Therefore, the accuracy of each classifier has improved. As shown in Table 7, the highest value is achieved by applying a soft voting ensemble machine learning classifier to reach 96.72%, followed by 96.14% by using SVC. Additionally, the accuracies achieved by other algorithms have increased to achieve 95.32%, 94.50%, 93.57%, and 90.18% by a hard voting ensemble, RF, LR, and MNB, respectively. Figures 7 and 8 show confusion matrices for the proposed ensemble algorithms and base ML classifiers with text augmentation. Furthermore, the text augmentation technique has affected the precision, recall, and F1-measure values of every class. Table 8 presents these values for all algorithms in this paper.

TABLE 9. Accuracy of algorithms with and without text augmentation.

Algorithm used in this paper	Accuracy before text augmentation	Accuracy after text augmentation
RF	88.55%	94.50%
MNB	83.64%	90.18%
SVC	88.78%	96.14%
LR	88.08%	93.57%
Proposed ensemble model (Hard voting)	89.01%	95.32%
Proposed ensemble model (Soft voting)	**90.42%**	**96.72%**

Moreover, Table 9 shows the effects of text augmentation on every algorithm on the dataset. The text augmentation technique has increased the accuracy of every algorithm. Without text augmentation, the highest accuracy has been achieved by a soft voting ensemble machine learning algorithm followed by a hard voting ensemble machine learning algorithm, then the SVC, RF, LR, and MNB classifiers. However, with text augmentation, there is a difference in which the highest accuracy has been achieved by soft voting ensemble machine learning algorithm followed by SVC, then hard voting ensemble machine learning algorithm, RF, LR, and MNB classifiers.

Additionally, the F1 scores for each nature class are different. However, without text augmentation, as Figure 9 shows, these values are approximately similar in the Test Code class and are higher than in other classes. This result could be because this class has less diversity of words than other classes. Moreover, the lowest F1 scores are in Network or Security class because of the limited number of bug reports in this class. The number of bug reports in this class without text augmentation is 242, which makes this class the lowest in terms of the size of its elements. Subsequently, less data is accessible for training the algorithms, and this lack affects the performance of the model in this class. Furthermore, as illustrated in Tables 6 and 8, Network or Security class has the highest precision values in most algorithms, which in this class these algorithms return more relevant nature of bugs than irrelevant ones.

V. DISCUSSION

This paper investigates the effect of machine learning algorithms in nature-based prediction of bug reports. In detail, a nature-based prediction model from bug reports based on ensemble machine learning algorithm is proposed. Additionally, text augmentation technique is used to improve prediction accuracy.

Four base machine learning algorithms have been used, Random Forest (RF), Logistic Regression (LR), Multinomial Naïve Bayes, and Support Vector Classifier (SVC) algorithms. Moreover, an ensemble machine learning model is proposed using hard voting and soft voting. The model achieves better accuracy than most existing models, using soft voting, it achieves 90.42% without text augmentation, and 96.72% with text augmentation.

It observed that our proposed ensemble machine learning model enhances the nature-based prediction accuracy than base machine learning classifiers.

Furthermore, it observed that there is a significant impact in prediction accuracy using text augmentation technique. However, the effect of text augmentation technique is different on each algorithm, as illustrated in table 9, without text augmentation, the highest accuracy has been achieved by a soft voting ensemble machine learning algorithm followed by a hard voting ensemble machine learning algorithm, then the SVC, RF, LR, and MNB classifiers. Whereas, with text augmentation, the highest accuracy has been achieved by soft voting ensemble machine learning algorithm followed by SVC, then hard voting ensemble machine learning algorithm, RF, LR, and MNB classifiers.

VI. CONCLUSION AND FUTURE WORK

This paper proposed a nature-based bug prediction component using an ensemble machine learning algorithm that consists of four base machine learning algorithms, Random Forest, Support Vector Classification, Logistic Regression, and Multinomial Naïve Bayes. The accuracy of the model is 90.42%. Moreover, it utilizes a text augmentation technique to increase accuracy. Therefore, the highest accuracy achieved by the proposed model increased to 96.72%. The proposed model predicts the nature of the bug from six bug categories, Program Anomaly, GUI, Network or Security, Configuration, Performance, and Test-Code. Future work will enhance this model by increasing the number of bug categories and recommending possible solutions for predicted bugs to reduce the maintenance time.

ACKNOWLEDGMENT

Deanship of Scientific Research (DSR) at King Abdulaziz University (KAU), Jeddah, Saudi Arabia, has funded this project, under grant no. (KEP-PhD-102-611-1443). The authors, therefore, acknowledge DSR for the financial support.

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 report repository," *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 on code 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 bug reports," 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 large scale 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 ACM SIGSOFT 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: A Web-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.

[26] X. Han, T. Yu, and D. Lo, "PerfLearner: Learning from bug reports to understand and generate performance test frames," in *Proc. 33rd IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Sep. 2018, pp. 17–28.

[27] P. E. Strandberg, T. J. Ostrand, E. J. Weyuker, W. Afzal, and D. Sundmark, "Intermittently failing tests in the embedded systems domain," in *Proc. 29th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2020, pp. 337–348, doi: 10.1145/3395363.3397359.

[28] Ankit and N. Saleena, "An ensemble classification system for Twitter sentiment analysis," *Proc. Comput. Sci.*, vol. 132, pp. 937–946, Jan. 2018.
[29] S. Kannan, V. Gurusamy, S. Vijayarani, J. Ilamathi, M. Nithya, S. Kannan, and V. Gurusamy, "Preprocessing techniques for text mining," *Int. J. Comput. Sci. Commun. Netw.*, vol. 5, no. 1, pp. 7–16, Oct. 2014.
[30] K. Rastogi. (Nov. 22, 2022). *Text Cleaning Methods in NLP*. Analytics Vidhya. Accessed: Jan. 23, 2023. [Online]. Available: https://www.analyticsvidhya.com/blog/2022/01/text-cleaning-methods-in-nlp/
[31] B. Gaye, D. Zhang, and A. Wulamu, "A tweet sentiment classification approach using a hybrid stacked ensemble technique," *Information*, vol. 12, no. 9, p. 374, Sep. 2021.
[32] S. Yang and H. Zhang, "Text mining of Twitter data using a latent Dirichlet allocation topic model and sentiment analysis," *Int. J. Comput. Inf. Eng.*, vol. 12, pp. 525–529, Jun. 2018.
[33] A. Rai and S. Borah, "Study of various methods for tokenization," in *Applications of Internet of Things*. Singapore: Springer, 2021.
[34] Y. Tian, D. Lo, and C. Sun, "DRONE: Predicting priority of reported bugs by multi-factor analysis," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Sep. 2013, pp. 22–28.
[35] I. Akhmetov, A. Pak, I. Ualiyeva, and A. Gelbukh, "Highly language-independent word lemmatization using a machine-learning classifier," *Computación y Sistemas*, vol. 24, no. 3, pp. 1353–1364, Sep. 2020.
[36] *Sklearn.feature_extraction.text.TfidfTransformer*. Scikit. Accessed: Jan. 24, 2023. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html
[37] *Sklearn.ensemble.randomforestclassifier*. Scikit. Accessed: Jan. 24, 2023. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[38] A. Bartosik and H. Whittingham, "Valuating safety and toxicity," in *ERA of Artificial Intelligence and Machine Learning*. New York, NY, USA: Academic, 2021, pp. 119–137.
[39] S. G. F. M. De, C. A. Netto, M. A. H. D. Andrade, M. M. A. D. Carvalho, and S. R. F. Da, "Engineering systems' fault diagnosis methods," in *Reliability Analysis and Asset Management of Engineering Systems*. Amsterdam, The Netherlands: Elsevier, 2022, pp. 165–187.
[40] R. Gandhi. *Naive Bayes Classifier*. Medium. Accessed: Jan. 29, 2023. [Online]. Available: https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c
[41] *Support Vector Machines*. Scikit. Accessed: Jan. 29, 2023. [Online]. Available: https://scikit-learn.org/stable/modules/SVM.html
[42] Y. Yang, "Introduction," in *Temporal Data Mining Via Unsupervised Ensemble Learning*. Amsterdam, The Netherlands: Elsevier, 2017, pp. 1–7.
[43] S. J. Simske, "Introduction, overview, and applications," in *Meta-Analytics: Consensus Approaches and System Patterns for Data Analysis*. San Diego, CA, USA: Elsevier, 2019, pp. 1–98.
[44] C. Kim. *Ensemble Learning-Voting and Bagging With Python*. Medium. Accessed: Jan. 30, 2023. [Online]. Available: https://medium.com/@chyun55555/ensemble-learning-voting-and-bagging-with-python-40de683b8ff0
[45] J. Brownlee. *How to Develop Voting Ensembles With Python*. Accessed: Jan. 30, 2023. [Online]. Available: https://machinelearningmastery.com/voting-ensembles-with-python/
[46] P. Tidke. *Text Data Augmentation in Natural Language Processing With Texattack*. Analytics Vidhya. Accessed: Jan. 31, 2023. [Online]. Available: https://www.analyticsvidhya.com/blog/2022/02/text-data-augmentation-in-natural-language-processing-with-texattack/
[47] N. Umasankar. *NLPAUG—A Python Library to Augment Your Text Data*. Analytics Vidhya. Accessed: Jan. 31, 2023. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/08/nlpaug-a-python-library-to-augment-your-text-data/
[48] M. Shafiq, "Identifying an effective set of attributes for machine learning based bug reports classification," M.S. thesis, Fac. Comput., Dept. Comput. Sci., Capital Univ. Sci. Technol., Islamabad, Pakistan, 2021.
[49] Z. Imran, "Predicting bug severity in open-source software systems using scalable machine learning techniques," M.S. thesis, Dept. Comput. Inf. Syst., Youngstown State Univ., Youngstown, OH, USA, 2016.
[50] N. Rahimi, F. Eassa, and L. Elrefaei, "An ensemble machine learning technique for functional requirement classification," *Symmetry*, vol. 12, no. 10, p. 1601, Sep. 2020.
[51] R. Chillarege, "Orthogonal defect classification," in *Handbook of Software Reliability Engineering*, M. R. Lyu, Ed. Piscataway, NJ, USA: IEEE Computer Society Press, 1996, pp. 359–399.

SHATHA ABED ALSAEDI received the B.Sc. degree in computer science from Umm AlQura University, Makkah, Saudi Arabia, and the M.Sc. degree in computer science from The University of Queensland, Brisbane, Australia. She is currently pursuing the Ph.D. degree in computer science with King Abdulaziz University, Jeddah, Saudi Arabia. She is a Lecturer with Taibah University, Yanbu, Saudi Arabia. Her current research interests include software engineering, machine learning, and natural language processing. In February 2017, she received the Dean's Commendation for Academic Excellence from the School of Information Technology and Electrical Engineering, The University of Queensland.

AMIN YOUSEF NOAMAN received the B.A. degree in computer science from the Faculty of Science, King Abdulaziz (KAU), Jeddah, Saudi Arabia, the M.A. degree in computer science from McGill University, Montreal, Canada, and the Ph.D. degree in computer science from Manitoba University, Winnipeg, Canada. He was a consultant in many companies and took part in founding several faculties and colleges with KAU. He was a Research Assistant with the Research Institute of Montreal, Canada; and an Assistant Professor, an Associate Professor, and a Professor with KAU. He occupied many administrative positions with KAU, including a Secretary of the Computer Sciences Council, Faculty of Computing and Information Technology; the Vice-Dean of development and technology with the Community College, the Deanship of Admission and Registration, the Deanship of Information Technology; the Dean of Admission and Registration; and the Vice-President of development with the Deputy Ministry of Education for Scholastic Affairs, Saudi Arabia. Currently, he is the Vice President of graduate studies and scientific research. His current research interests include big data, data warehousing, data mining, bioinformatics, smart cities, and e-learning.

AHMED A. A. GAD-ELRAB received the B.S. degree in computer science from the Faculty of Science, Alexandria University, Egypt, in 1999, the M.S. degree in computer science from the Faculty of Science, Cairo University, Egypt, in 2008, and the Ph.D. degree from the Nara Institute of Science and Technology (NAIST), Japan, in 2012. Currently, he is an Associate Professor with the Department of Computer Science, Faculty of Computing and Information Technology, King Abdul-Aziz University, Jeddah, Saudi Arabia. He is also an Associate Professor of ubiquitous and mobile computing with the Department of Mathematics, Faculty of Science, Al-Azhar University, Cairo, Egypt. His current research interests include software engineering, cloud computing, mobile computing, the Internet of Things applications, smart homes, data science, sensor networks, dynamic distributed systems, big data, and mobile crowd sensing. He received the NAIST Best Ph.D. Student Award, in March 2012; the Outperformance Award from the Graduate School of Information Science, NAIST, in March 2012; and the 2011 IPSJ Yamashita Memorial Award (given to only one or two papers among all papers presented in one year in each IPSJ SIG, Japan).

FATHY ELBOURAEY EASSA received the B.Sc. degree in electronics and electrical communication engineering from Cairo University, Egypt, in 1978, and the M.Sc. and Ph.D. degrees in computers and systems engineering from Al-Azhar University, Cairo, Egypt, in 1984 and 1989, respectively, joint supervision with the University of Colorado at Boulder, Boulder, CO, USA. He is currently a Full Professor with the Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Saudi Arabia. His current research interests include agent-based software engineering, the IoT security, software engineering, big data management and security, distributed systems security, and exascale systems testing.