e-Informatica Software Engineering Journal, Volume 15, Issue 1, 2021, pages: 29–45, DOI 10.37190/e-Inf210102

A Deep-Learning-Based Bug Priority Prediction Using RNN-LSTM Neural Networks

Hani Bani-Salameh[*], Mohammed Sallam[*], Bashar Al shboul[*]

[*]*Department of Software Engineering, The Hashemite University*

hani@hu.edu.jo, mohammed.yasser@intix.net, bashar.alshboul@hu.edu.jo

https://orcid.org/0000-0002-2962-9449, https://orcid.org/0000-0003-4765-4653, https://orcid.org/0000-0002-1814-645X

Abstract

Context: Predicting the priority of bug reports is an important activity in software maintenance. Bug priority refers to the order in which a bug or defect should be resolved. A huge number of bug reports are submitted every day. Manual filtering of bug reports and assigning priority to each report is a heavy process, which requires time, resources, and expertise. In many cases mistakes happen when priority is assigned manually, which prevents the developers from finishing their tasks, fixing bugs, and improve the quality.
Objective: Bugs are widespread and there is a noticeable increase in the number of bug reports that are submitted by the users and teams' members with the presence of limited resources, which raises the fact that there is a need for a model that focuses on detecting the priority of bug reports, and allows developers to find the highest priority bug reports.
This paper presents a model that focuses on predicting and assigning a priority level (high or low) for each bug report.
Method: This model considers a set of factors (indicators) such as component name, summary, assignee, and reporter that possibly affect the priority level of a bug report. The factors are extracted as features from a dataset built using bug reports that are taken from closed-source projects stored in the JIRA bug tracking system, which are used then to train and test the framework. Also, this work presents a tool that helps developers to assign a priority level for the bug report automatically and based on the LSTM's model prediction.
Results: Our experiments consisted of applying a 5-layer deep learning RNN-LSTM neural network and comparing the results with Support Vector Machine (SVM) and K-nearest neighbors (KNN) to predict the priority of bug reports.
The performance of the proposed RNN-LSTM model has been analyzed over the JIRA dataset with more than 2000 bug reports. The proposed model has been found 90% accurate in comparison with KNN (74%) and SVM (87%). On average, RNN-LSTM improves the F-measure by 3% compared to SVM and 15.2% compared to KNN.
Conclusion: It concluded that LSTM predicts and assigns the priority of the bug more accurately and effectively than the other ML algorithms (KNN and SVM). LSTM significantly improves the average F-measure in comparison to the other classifiers. The study showed that LSTM reported the best performance results based on all performance measures (Accuracy = 0.908, AUC = 0.95, F-measure = 0.892).

Keywords: Assigning, Priority, Bug Tracking Systems, Bug Priority, Bug Severity, Closed-Source, Data Mining, Machine Learning (ML), Deep Learning, RNN-LSTM, SVM, KNN

Submitted: 03 July 2020; Revised: 07 November 2020; Accepted: 23 December 2020; Available online: 08 February 2021

1. Introduction

Software projects (both open and closed source) get an overwhelming number of bug reports, and the presence of bugs usually affects reliability, quality, and cost management of software. In practice, it is impossible to have a bug-free software unless the software is implemented carefully and developers can quantify software behaviors as being a bug or not [1].

Bugs are prevalent, and many software projects are delivered with bugs. To address these bugs and to improve the quality of the released products, bug tracking systems such as JIRA and Bugzilla allow users and team members to report bugs [2].

Bug tracking systems help to predict the progress of a milestone-based on bug reports raised. They allow users to add stories (functional requirements) and divide them into tasks, as well as preparing bug reports and test suites [2].

Developers and testers can create new bug reports, monitor the state of bug reports as well as any update on existing bug reports. Bug reports progress through a series of states, where the bug reports begin when the bug is found and ends when the bug reports are closed [1].

Bug reports may then be used to direct the software corrective maintenance behavior and contribute to creating more stable software systems. Prioritizing software bug reports can help to handle the bug triaging process, and allows developers to prioritize and fix important reports first [2]. Developers are often receiving numerous bugs reports and may fail to fix it due to different constraints including time. The process of prioritizing bug reports is manual and is time-consuming. Thus, there is a need to develop a bug's priority prediction model that helps to automate the priority's prediction process.

The model helps to (1) improve accuracy and effectiveness in predicting the priority of the bug reports, (2) improve efficiency by reducing the time spent during manual priority prediction, and (3) reduce the cost of assigning incorrect priority.

This article proposes a framework that used a dataset extracted from five closed-source projects containing more than 2000 bug reports (provided by JIRA). Also, it uses different algorithms, namely RNN-LSTM, SVM, and KKN, to predict the priority and compare the accuracy results.

The rest of this paper is organized as follows. The rest of Section 1 provides background about the bug reports lifecycle and machine learning (ML) and its relationship with software bug problems. Section 2 presents the related works. Section 3 presents the detailed description of the proposed approach. The results of the study are presented in Section 4. Section 5 discusses the priority prediction tool. The possible threats to the validity of our work were listed in Section 6. Finally, Section 7 presents the conclusion of the research work along with future work directions and enhancement.

1.1. Bug reports lifecycle

Bug reports go through a cycle during their lifetime. This article divides the life cycle of the bug reports into five states: *Open*, *InProgress*, *Resolved*, *Closed*, and *Reopened*. These phases are described hereunder (see Fig. 1).

When a tester posts a bug, a bug report is **opened** and logged in to the tracking system, the status is set to OPEN. After that, the leader approves that the bug exists and assigns the bug to the appropriate developer. Once the developer starts analysis and works on fixing the bug, the status set to **INPROGRESS**.

If a bug is posted twice, the status is set to **CLOSED** with a resolution *duplicate*. If the developer feels that the bug is not logical or incompatible with the specific user experience, the status is set to **CLOSED** with resolution *will not do*. If the bug is not reproducible (all attempts to reproduce this bug have failed, or insufficient information was available to reproduce the bug), then the status is set to **CLOSED** with resolution *cannot reproduce*.

Once the developer fixes the issue and verifies the changes, the status is set to **RESOLVED**. After fixing the bug, testing is pending and the tester either confirms the change or re-test the

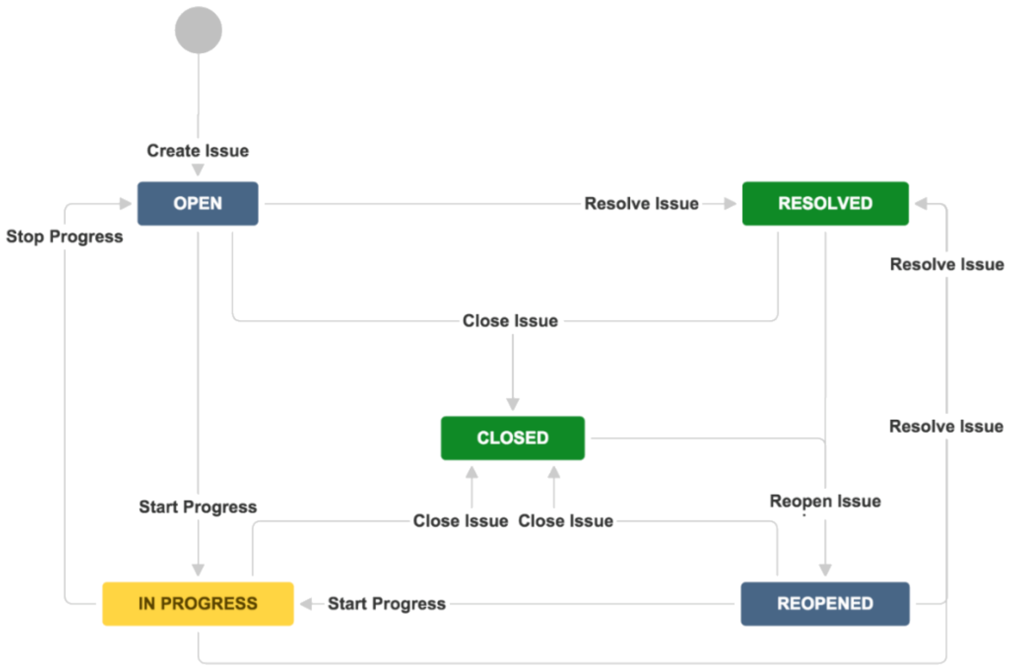


Figure 1: Lifecycle of bug reports

changes to make sure that the bug is no longer exists in the software. Next, the status is set to **CLOSED** with the decision done. If the bug still exists and not resolved, the status is set to **REOPEN**.

Finally, when the software delivery date (deadline) reaches and low priority bugs are not fixed, they must be moved to the next release by the product owner, and the status remains **OPENED**.

1.2. Bug reports contents

A bug report contains information on how the bug could be reproduced and the information that can help in its debugging and tracing. A bug report includes a set of factors like *summary*, *descriptions*, *report id*, *project name*, *priority*, *environment*, *attachment*, *assignee*, *reporter*, *created date*, *status*, *fix version*, and *component*. Table 1 shows the defined factors.

Table 1: Summary of bug report fields

Field	Description
Summary	A brief one-line summary of the bug.
Descriptions	Details including test step, actual result, and expected result to reproduce this bug
Report ID	A unique identifier for this bug
Project Name	The parent project to which the bug belongs.
Priority	How quickly a bug should be fixed and deployed (e.g., Low, Medium, and High)
Environment	The environment in which the issue occurred (e.g., production, pre-production, staging, and development.)
Attachment	Documents, screenshots, and other elements that can help in identifying and fixing bugs.
Assignee	A person who created the bug (e.g., QA).
Reporter	A person who is responsible for fixing the bug (e.g., QA, scrum master, and owner).
Created Date	Date when a bug is submitted.
Status	The stage the bug is currently in during the lifecycle (workflow).
Fix Version	Project version(s) that contains the bug.
Component	Component(s) to which the bug relates (e.g., Android, IOS, and Backend (DB)).

2. Related works

This article focuses on related works and studies that are mainly related to machine learning (ML) and the techniques were applied to assigning bugs priority and prediction. This section introduces recent studies and literature that are related to bugs' priority.

2.1. Bug reports assignment

Anvik et al. [3] introduced a machine learning method that classifies appropriate developer names to resolve the report based on classifying and reporting bug using accuracy and recall. Applying their method on Firefox and Eclipse, they achieved +50% accuracy.

Wang et al. [4] address three limitations of the supervised bug fix approaches and propose an unsupervised method for assigning bugs for developers based on their involvement ("activeness score") in the project. The result of experiments showed that FixerCache gives better accuracy when compared with the supervised approaches, and it achieves prediction accuracy up to 96.32% and diversity up to 91.67% in Eclipse and Mozilla datasets.

Recently, Mani et al. [5] proposed an algorithm using a deep-bidirectional recurrent neural network (DBRNN-A) model. The model is for a specific software bug reports classifying an adequate developer depending on the title and characteristics of the bug reports using naive Bayes, cosine distance, SVM, and softmax. Experiments on bug reports from software projects (e.g., Google Chromium, Mozilla Core, and Mozilla Firefox). It showed a precision of 47% for Google Chromium, 43% precision for Mozilla Core, and 56% precision for Mozilla Firefox.

2.2. Bug priority prediction

Prioritizing bug reports is not an easy task. Only a small percentage of bug reports are extremely impactive reports (e.g., according to Ohira et al. [6] less than 1% of Ambari bug reports are absent in the dataset).

2.2.1. Traditional approaches to bug priority prediction

Tian et al. [2, 7] suggested an automated classification method to predict the priority of bug reports. They used a machine learning (ML) algorithm to prioritize bug reports and achieved an average F-measure of 209%. The dataset of the bug reports is usually unbalanced according to the low number of high impact bugs in the project.

Umer et al. [8] proposed an emotion-based automated priority prediction approach. Their approach combines the NLP techniques and ML algorithms. It allows team members to assign appropriate priority level bug reports in an automated manner. The results suggest that the proposed approach outperforms the state-of-the-are and it improves $F1$-score by more than 6%.

Mihaylov [9] conducted a study that aims to examine the behavior of NNs and predict the priority of bug reports. Their focus was to analyze the importance of adding numerical features to textual features by combining different kinds of NNs. The results suggest that adding numerical features to textual features improves the accuracy of priority classification. The results show that the priority classification improves the accuracy of about 85.5%.

Choudhary et al. [10] introduce an ANN technique used to develop prediction models for several Eclipse versions that set priority levels based on the textual, temporal, relevant report, author, severity, and the product.

Yu et al. [11] proposed an enhanced ANN--based system to predict the priorities of five different product bugs identified by an international health-care company. The threefold cross-validation tests suggest that the alternative approach is better in terms of precision, recall, and F-measure.

Jaweria Kanwal [12] proposed an ML-based recommender to automatically prioritize reported bugs. They used SVM to train a classification-based approach on Eclipse bug reports. The evaluation of the proposed approach used precision, retrieval, and F-measure to set the priority of the automatic defect.

Lin et al. [19] applies both of SVM and C4.5 classifiers on different fields (e.g., bug type, submitter, phase-ID, module-ID, and priority). They used a dataset with 2,576 bug reports. Their models achieve the accuracy of up to 77.64%.

Sharma et al. [13] proposed a priority prediction approach using SVM, NB, KNN, and neural networks. The proposed approach allows to predict the priority of the bug reports. The results showed that the accuracy of the used machine learning techniques in predicting the priority of bugs' reports within the project is found above 70% except NB technique.

Alenezi and Banitaan [14] proposed an approach to predict bugs' priority prediction. They used different ML techniques NB, Decision Tree, and Random Forests. The results show that the proposed approach is feasible in predicting the

Table 2: Summary of machine learning based bug priority approaches available in literature

Paper(s)	Performance	Features Used	Classifier(s)
		Traditional Bug Priority Prediction	
[7]	improves the average of F-measure by a relative improvement of 58.61%	temporal, textual, author, related-report, severity, product	Drone, SVM, NBM,
[8]	improves F-score by more than 6%	summary	NLP + ML algorithm
[9]	accuracy = 85.5%	sentiment and textual analysis	MLP, CNN, LSTM
[10]	both algorithms are efficient	temporal, textual, severity, product, component	MLP, NB
[11]	suggested improvement in terms of precision, recall, and F-measure	milestone, category, module, main workflow, function, integration, frequency, severity, and tester	Rnhanced ANN, Bayes
[12]	SVM is better	categorical, summary, long description	SVM, NB
[13]	above 70% except NB		SVM, NB, KNN
[14]	F-measure values (Random Forest = 0.611, Decision Trees = 0.603, NB = 0.593)	component, operating system, severity	RF, DT, NB
[15]	accuracy = 75%	–	Decision Trees (DT), Random Forest(RF)
		Deep Learning in Bug Priority Prediction	
[5]	improvement = 12–15%, accuracy = 37–43%	title, description	softmax, SVM, MNB, cosine distance based machine
[16]	accuracy = 56–88%	title, component, priority, product	NB, TF-IDF with SVM, fastText, and DeepTriag
[17]	$F1$-measure improved by 14% and AUC by 7%	–	CNN, RNN-LSTM, and DP-ARNN
[18]	$F1$-measure improved by 7.9%	–	CNN, LSTM), Multinomial NB (MNB), RF
Our approach	accuracy = 90.8%	*component*, *summary*, *assignee*, and *reporter of bug reports.*	LSTM, SVM, KNN

priority of bug reports. Also, the study shows that Random Forests and Decision Trees beat NB.

Others [15] They proposed an approach that constructs multiple decision trees based on existing datasets and features, which selects the best decision trees to measure the new bugs' priority and severity. They proposed the applicability of random forest (RF) for bug reports analysis. Results showed that RF yields 75% as an accuracy score.

2.2.2. Deep learning approaches to bug priority prediction

Mani et al. [5] propose a a bug report representation algorithm using deep-bidirectional RNN network model (DBRNN-A). They chose two features as input for the classification (title, description of the issues). They used bug reports from different projects such as Chromium, Mozilla Core, and Mozilla Firefox. The result shows that DBRNN-A achieves an improvement of 12–15% and performance between 37–43% when compared to other classifiers. Lyubinets et. al [16] present a model to label bugs reports using RNNs. The achieved accuracy were 56–88%.

Fan et al. [17] proposed a deep learning-based method called DP-ARNN, to help predict prospective code defects. They used the attention mechanism to capture important features that might help improve the defect prediction performance. They made use of seven open-source projects. Results indicated that DP-ARNN improves the state-of-the-art traditional methods of software defects prediction where $F1$-measure improved by 14% and AUC improved by 7%.

Ramay et al. [18] proposed a deep neural network-based approach for bug reports severity. They evaluated their model on the history-data of bug reports. The results showed that there is an improvement in the F-measure by 7.90%, which indicates that the approach outperforms the state-of-the-art approaches.

The above mentioned closely related works on bug priority are summarized in Table 2.

These studies have focused on using on both of deep learning and traditional classification algorithms such as C4.5, Bayesian, MLP, and Support Vector Machine (SVM). Our work presents an approach to predict and assign bugs' priority level using deep learning. The proposed approach use of LSTM and outperforms the other classifiers where accuracy improved by 90.8%.

3. Proposed approach

Given a dataset of bug reports from closed-source software projects, this study uses RNN-LSTM neural networks to detect and prioritizes bug reports. The process of assigning the priority level for bug reports consists of two phases (see Fig. 2). This section briefly explains the process phases.

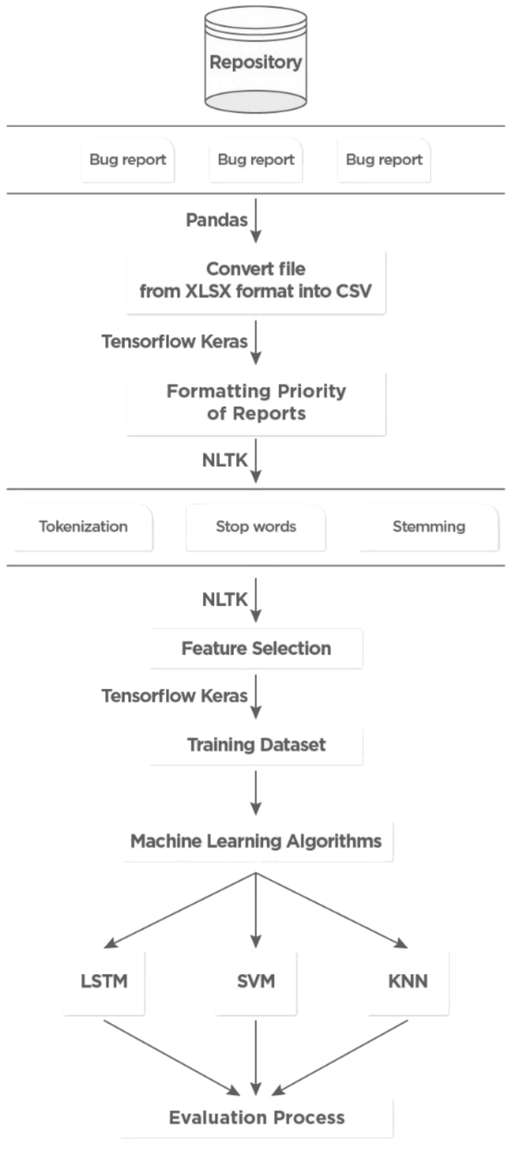


Figure 2: Proposed framework

Phase 1: involves data collection, formatting priority of the bug reports, and text preprocessing (tokenization, stop words, and stemming).

Phase 2: involves feature selection, dataset training, applying ML algorithms (LSTM, SVM, and KNN), and finally evaluation process.

3.1. Data collection

As mentioned earlier, this study used a dataset that was extracted from the JIRA bug tracking system using the INTIX DWC company dashboard [25]. The datasets consist of data from five closed-source projects and containing more than 2000 bug reports. Past studies [3, 4, 12, 26–28] used common datasets extracted from Bugzilla [29] system that are related to Eclipse and Mozilla.

JIRA dataset consists of 17 columns. The factors are summary, description, bug id, status, project name, project lead, priority, resolution, assignee, reporter, created date, resolved date, component, environment, sprint, attachment files, and comments.

In this work, we used a specific number of factors from the chosen dataset. The factors that are considered as the most appropriate to predict the priority level (high, medium, and low): *component*, *summary*, *assignee*, and *reporter of bug reports.*

Table 3 shows the bug reports in the closed--source projects included in the used dataset, which are *Martix*, *Hashfood*, *Tazaj*, *Workspaces*, and *Maharah.* The project with the highest number of bug reports is *Hashfood.* The bug reports are divided into three levels of priority (high, medium, and low). The number of bug reports with medium priority is higher in each dataset compared to the number of low and high priorities.

3.1.1. Labeling priority of reports

The priority of bug reports was labeled using the *to_categorical* function from the TensorFlow Keras library [30]. The used labels are 0 and 1, where 0 refers to the high priority and 1 refers to the low priority of bug reports.

3.1.2. Text preprocessing

Text preprocessing is applied using the Natural Language Toolkit library [31]. This is performed by practicing Python programming using PyCharm [32].

This section gives a brief definition of each activity.

– **Tokenization:** the process of splitting text into sentences, words, and clauses. It replaces all the punctuations with blank spaces, re-

Table 3: High, medium, low, and unselect priority levels in each project dataset

Projects name	High	Medium	Low	Unselect
Martix [20]	207	489	135	38
Hashfood [21]	453	659	95	0
Tazaj [22]	80	169	18	0
Workspaces [23]	143	186	9	0
Maharah [24]	55	61	9	0

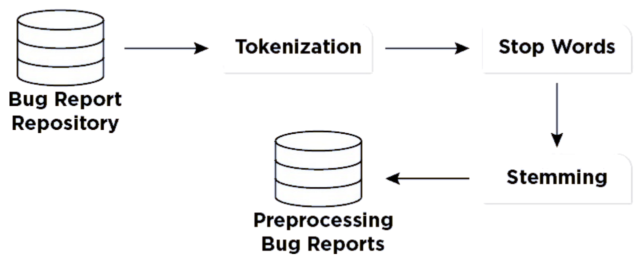


Figure 3: Text preprocessing activities

Table 4: Example illustrates the effect of preprocessing activities

Original Description	Crashed when clicking on order details “Consumer application”
Tokenization	Crash when I click on order details consumer application
Stop Words	Crashed click order details consumer application
Stemming	Crash click order detail consumer application

Table 5: The top-30 keywords based on their frequency (sorted using NLTK)

Rank	Keyword	Rank	Keyword	Rank	Keyword
1	IOS	11	search	21	back
2	Android	12	seller	22	login
3	Screen	13	click	23	logo
4	app	14	App	24	account
5	incorrect	15	Api	25	network
6	message	16	backend	26	payment
7	product	17	chat	27	google
8	user	18	button	28	service
9	order	19	mobile	29	server
10	error	20	design	30	web

Table 6: Keywords classified based on the priority level

Keywords	Count of frequency	Priority level	Keywords	Count of frequency	Priority level
crash	186	high	color	29	low
error	159	high	inconsistent	22	low
icon	110	low	layout	21	low
photo	108	low	avatar	20	low
tab	65	low	placeholder	20	low
image	101	low	doesn’t Work	16	high
menu	53	low	ux	16	low
design	52	low	toolbar	15	low
logo	47	low	textview	9	low
label	45	low	hot fix	9	high
title	34	low	failure	8	high

moves all the nonprintable escape characters, and converts all the words to lowercase [7].

– **Stop word removal:** prepositions, articles, conjunctions, verbs, pronouns, nouns, adjectives, and adverbs, which has no meaning in NL processing [7].
– **Stemming:** is the process for reducing words to their stem or root. All words with a common stem are replaced. For example, words like “take”, “takes”, “took”, and “taking” can be replaced with a single word as “take” [7].

Table 4 illustrates the preprocessing activities using Natural Language Toolkit (NLTK) [31].

3.1.3. Feature selection

Text preprocessing generates a large set of features that are still costly to be processed using the proposed machine learning algorithms. Thus, it is important to decide what features of the input are relevant.

Various techniques have been proposed to derive relevant features (terms/keywords) from the bug reports. This research used the NLTK library [7] as a features’ selection technique to reduce the number of input features and help improve the performance.

Features are derived from the bug reports to provide the most important words that impact the priority level, then the words are ranked highest to lowest based on the frequency of the word (see Table 5).

Also, manual analysis was applied and identified the strongest set of keywords that refer to the low and high priority levels of the bug reports (see Table 6).

3.2. Evaluation metrics

The performance and effectiveness of the classification algorithms were evaluated using well-known metrics such as *precision*, *accuracy*, *recall*, *F-measure*, and *improvement* [18].

Also, these metrics were used to evaluate the performance of the proposed approach to the priority bug reports. The metrics present the *precision*, *accuracy*, *recall*, and *F-measure* of the proposed approach in assigning priority of the bug reports. Following is a description of the used metrics.

Accuracy is the percentage of correctly predicted observation to the total, which is considered as an important performance measure when using asymmetric datasets that present when false positive and false negatives are the same value [33]. Accuracy can be measured using the following formula:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

Precision is the ratio of correctly predicted positive to the total predicted positive. The percentage of priority bug reports was predicted, and then considered precision for the high and low level of priority [33]. Precision can be measured using the following formula:

$$Precision = \frac{TP}{FP + TP} \tag{2}$$

Recall (Sensitivity) is the ratio of correctly predicted positive to the total observation in the same class. The percentage of all high-priority and low-priority bug reports that are correctly predicted [33]. Recall can be measured using the following formula:

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

F-measure means the average accuracy and recall taking into account false positives and false negatives. *F*-measure is more effective than accuracy, especially if the data distribution is unbalanced. If false positives and false negatives have the same results, this means that the accuracy is more effective [33]. *F*-measure can be measured using the following formula:

$$F\text{-measure} = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4}$$

Improvement considers calculating the improvement between selected classification algorithms [34]. Improvement can be measured using the following formula:

$$Improvement = \frac{(F\text{-measure}_{\text{LSTM}}) - (F\text{-measure}_{\text{KNN}})}{F - \text{measure}_{\text{KNN}}} \tag{5}$$

Also, to measure the quality of the classifiers, we calculated Mathews Coefficient Correlation (MCC).

$$MCC = \frac{(TP*TN)-(FP*FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{6}$$

3.3. Machine learning algorithms

The proposed approach is compared with some existing ML algorithms. We used three different machine learning algorithms to predict the bug reports' priority. Theses algorithms are Long Short-Term Memory (LSTM), Support Vector Machine (SVM), and *K*-nearest neighbors (KNN).

3.3.1. Long short-term memory (LSTM)

RNN-LSTM is an example of supervised learning used in deep learning, which uses history measurements to generate bandwidth prediction

Table 7: Factors considered from the bug reports

Field name	Field description	Field value
Component	Bug component(s) to which this bug relates.	Android, IOS, Backend (DB).
Summary	A brief one-line summary of the bug.	String(crash, failure, design, UX, or UI).
Assignee	A person who created a bug	Senior, Junior, and a fresh graduate.
Reporter	A person who is responsible to fix the bug.	QA, developer, scrum master, product owner.

Table 8: Output variable

Field name	Field description	Field value
Priority	How quickly the bug should be fixed and deployed?	Low/High

and remembers the information for long periods. It can learn to transform input data into a preferred response and is widely used for prediction problems [35]. RNN-LSTM can remember past events that are seen and forget unimportant data. This happens through different activation function layers called gates. The Internal Cell State presents the relevant information that was selected to be saved. RNN-LSTM is a type of a Recurrent Neural Network (RNN) that uses past events to inform future ones [35–38].

In this research, we implemented a Python code for the LSTM neural network with five hidden layers feed-forward to predict the priority of the bug reports. The number of hidden layers has been selected to achieve the best performance. It includes a cell that saves relevant information which has an impact on the priority level. The model helps to assign an appropriate priority level of bug reports [38].

3.3.2. Support vector machine (SVM)

SVM is a supervised machine learning model that applies classification on two-group classification problems. It can be used for classifications, regression, and outliers' detection. The objective of applying SVMs is to classify the dataset space by finding the best line or hyperplane [39–41]. SVM is implemented using a Python code, mainly using Sklearn.svm library [42, 43].

3.3.3. *K*-nearest neighbors (KNN)

KNN is a supervised machine learning algorithm that can be used to solve both classification and regression problems. Using KNN, the input variables consist of the k closest training examples in the dataset. Predicting the output depends on whether k-NN is used for classification or regression problems [44, 45]. KNN is implemented using a Python code, mainly using Sklearn.neighbours library [42].

3.4. Building the LSTM neural network

As mentioned earlier, Python was used to implement the LSTM neural network with five hidden layers feed-forward to predict the priority of bug reports.

3.4.1. Input variables

The input variables were selected to predict the priority level by considering a set of factors (indicators). The factors are component name, summary, assignee, and reporter of the bug reports (see Table 7).

3.4.2. Output variables

In this study, *Priority* is the output variable used in the ML algorithms to be predicted (see Table 8).

3.5. Supervised training of LSTM

To train the LSTM neural network, we are using a python library called *TensorFlow* [46] that divides the datasets into training and test sets in the ratio 8:2. The datasets were converted from the .xlsx format into .cvs format using the panda's library [47] to make the module faster.

The priority data is converted into [0, 1] using the *to_categorical* function from the TensorFlow Keras library [46]. The assigned values to the priority of bug reports are (0 = high priority, 1 = low priority).

The architecture of LSTM units was trained using Adam algorithms and the Mean Square Error loss function. Adam algorithm has been used in our research instead of the traditional algorithms to update network weights based on training data [48]. The main benefits of *Adam* algorithms are computationally efficient and suitable for handling problems with large data.

The learning rate variable is set to 0.001 and it decays every 5 epochs and drops-out in each layer in 0.2 to remove any loss value in validation split. In this article, the model has been trained with 100 sequences per batch and the count of the batches is 64 with patience from 3 samples.

4. Experimental results and discussion

The performance of the proposed LSTM model was evaluated by the experiment on a dataset extracted from five different projects, which are *Matrix*, *Hashfood*, *Tazaj*, *Workspaces*, and *Maharah*, which have a different number of priority bug reports, as shown earlier in Table 4.

The proposed approach is compared with the existing ML algorithm. Three ML algorithms were utilized on the selected dataset. The evaluation was performed with *LSTM*, *KNN*, and *SVM* after defining that the test size is equal to 0.20 of our dataset, then the bug reports were selected randomly. This is done using the *train_test_split* Python library to split datasets into a random train and test subsets [49]. The model was trained and tested with more than 2000 bug reports.

This rest of this section presents the results of the experiment that was conducted to validate the proposed model and answers the research question.

4.1. Research question

This work investigates the following research question to evaluate the proposed framework:

"Does the proposed approach outperform the other machine learning algorithms in predicting and assigning bug priority? Does the proposed approach improve the accuracy of assigning priority levels of bug reports?"

The research question compares the selected deep neural network (LSTM) against other alternatives as shown in sections 4.1–4.3. Also, it investigates the performance improvement of the proposed approach as shown in Sections 4.4 and 4.5.

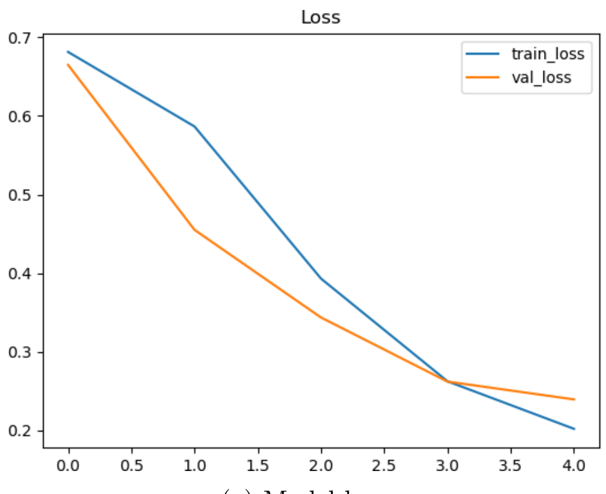
4.2. LSTM neural network

The experiments performed on LSTM Neural Network after training epoch. The LSTM recurrent neural network model was developed in Python using the Keras deep learning library [30].

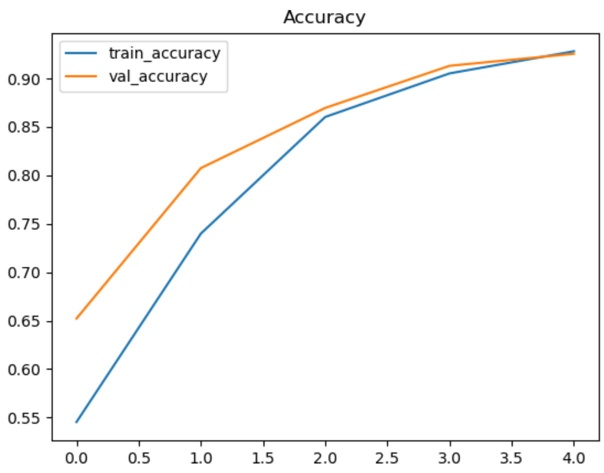
Accuracy and loss in Keras model for validation data could be changed with different cases. When every epoch increases, the loss becomes lower and the accuracy becomes higher. With Keras validation loss and accuracy, the following cases may occur [50]:

- **The model is not learning (cramming values):** when validation loss starts increasing, validation accuracy starts decreasing.
- **Overfitting:** both of validation loss and validation accuracy start increasing.
- **The model is learning probably:** validation loss starts decreasing, and validation accuracy starts increasing.

As shown in Figures 4(a) and 4(b), this research defined the loss and accuracy functions which are considered as a return to the difference between the training and testing data (predicted and actual outcome). Then we calculated the accuracy, precision, recall, and F-measure.



(a) Model loss



(b) Model accuracy

Figure 4: Comparison between training and validation(loss and accuracy)

Figures 4(a) and 4(b) illustrate that the model has higher training accuracy and lower validation accuracy, thus it is learning probably. The training loss is decreasing, which means that the model is learning to recognize the training set. Also, the model is a good fit because training loss is slightly higher than validation loss.

4.3. Support vector machine (SVM)

This section presents the results when SVM was applied to datasets extracted from closed-source projects. Table 9 shows the performance results of the SVM model based on the level of priority. Based on the High priority, the metrics values are (F-measure = 0.87, *Recall* = 0.88, and *Precision* = 0.85). Based on the Low priority, the metrics values are (F-measure = 0.86, *Recall* = 0.85, and *Precision* = 0.88).

Table 9: Performance metrics results from applying SVM

Priority level	Precision	Recall	F-measure
High	0.85	0.88	0.87
Low	0.88	0.85	0.86

4.4. *K*-nearest neighbors (KNN)

Based on the performance results of KNN, the metrics values are *Accuracy* = 0.741, F-measure = 0.740, *Recall* = 0.742, and *Precision* = 0.740.

Table 10 shows the evaluation results of the three algorithms. It shows the performance of LSTM, SVM, and KNN.

Table 10: A comparison between performance results from applying LSTM, SVM, and KNN

	LSTM	SVM	KNN
Accuracy	0.898	0.865	0.741
F-measure	0.892	0.865	0.742
Recall	0.897	0.865	0.741
Precision	0.876	0.866	0.743
MCC	0.796	0.732	0.485

Figure 5 illustrates the performance differences between the three ML algorithms.

4.5. Comparison between LSTM, SVM and KNN results

The results of our experiments indicate that the proposed framework based on LSTM Neural Network correctly predicts the priority of the bug reports and the performance can be significantly increased compared with both SVM and KNN as shown in Figure 5.

Based on Table 11 and Figure 5, we make the following observations:

– The proposed approach obtains a slight improvement in performance. The LSTM improvement was calculated and compared with the other selected algorithms SVM and KNN.
– F-measure results show a 3% improvement for LSTM compared with SVM. Also, it shows

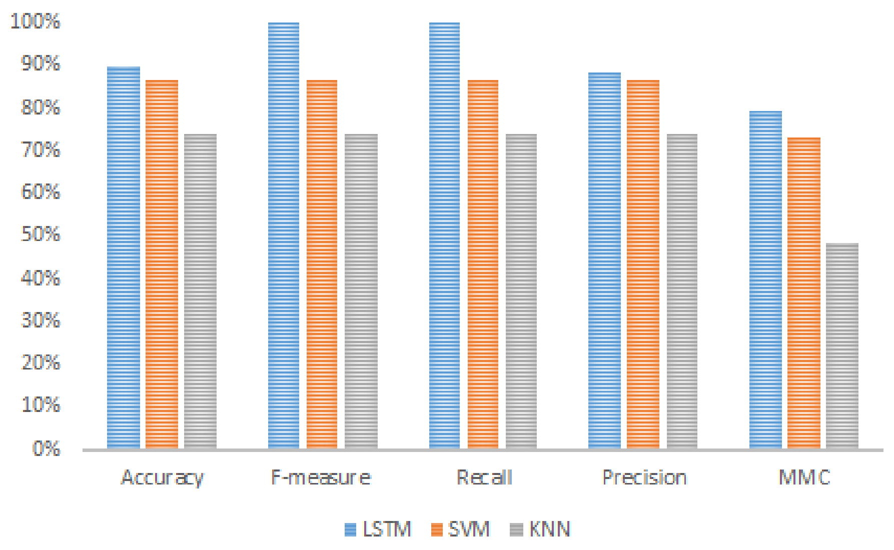


Figure 5: Comparison between LSTM, SVM, and KNN

Table 11: LSTM Improvement compared to SVM and KNN

	SVM	LSTM	Improvement		KNN	LSTM	Improvement
F-measure	0.865	0.892	3%	F-measure	0.742	0.892	15.2%
MCC	0.732	0.796	6.4%	MCC	0.485	0.796	31.1%

a 15% improvement for LSTM compared with KNN.

– MCC values improved by 6.4% compared to SVM and by 31.1% compared to KNN, which show that LSTM outperforms the other algorithms in detecting and assigning the bugs priority.

4.6. LSTM, SVM, KNN – Output quality comparison

To compare the selected algorithms, this study summarizes and compares the performance of each classifier by calculating the area under the ROC curve (AUC).

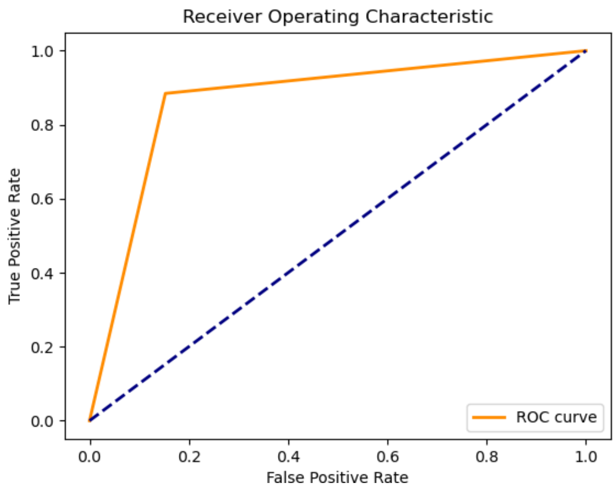


Figure 6: ROC curve for SVM

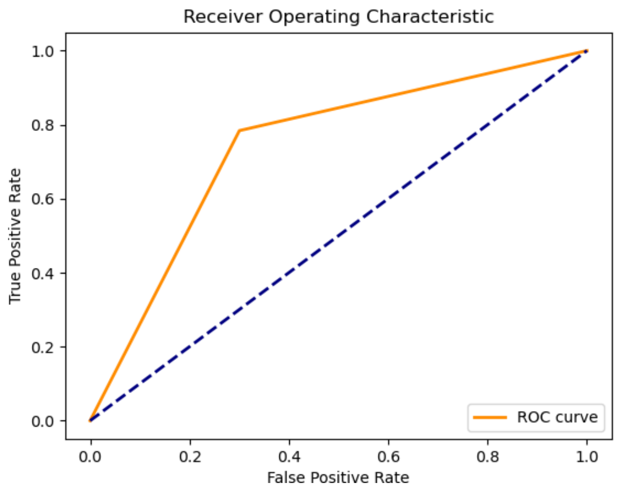


Figure 7: ROC curve for KNN

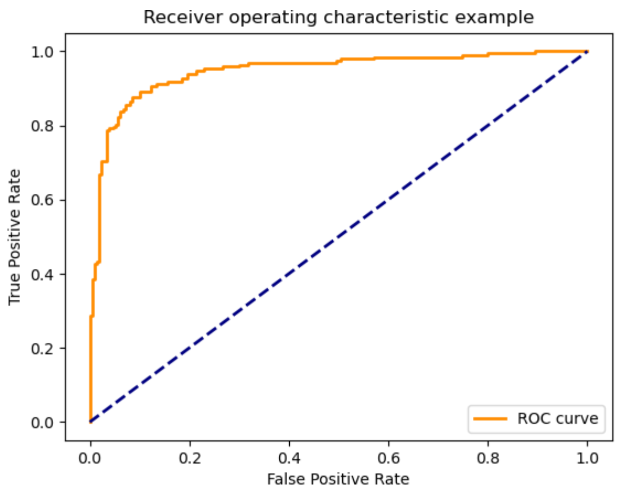
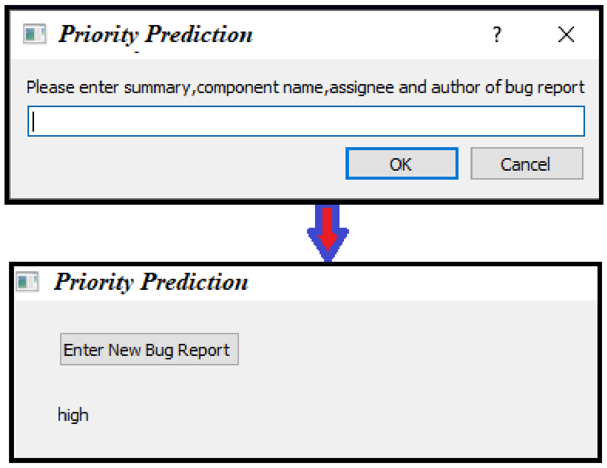


Figure 8: ROC curve for LSTM

Results show that LSTM is with better AUC, which is an effective measure of sensitivity and specificity (a measure of predictive accuracy). The AUC values for LSTM, SVM, and KNN are 0.95, 0.87, and 0.74, respectively (see Figures 6–8).

5. Bugs' priority prediction tool

Assigning priority to bugs' reports may play an important role in improving the bug triaging process which is an important process in software maintenance.



Please enter summary,component name,assignee and author of bug report

high

Figure 9: Predicting priority level for a bug report

This article introduces a tool that helps developers and team to predict and assign priority for bugs' reports. This tool was built using pyqt5.qt widgets [51]. It allows them to enter the input (RNN-LSTM input features) as a single comma separated statement (component name, summary, assignee, and author). Then, the neural network predicts the priority of the report. Figure 9 shows an example of the labeling panel in the proposed tool.

6. Threats to validity

Like any research, some factors may affect the performance of the proposed approach. The threats to the validity of our study are as follows.

The internal validity relates to the adoption of LSTM and not the other algorithms. We chose LSTM since others proved it effective for text classification [52, 53]. Also, the results are verified to avoid any errors.

External validity makes it difficult to generalize the results. As mentioned earlier the used dataset extracted from bug reports related to five closed-source projects. Using datasets from other projects, it is not sure to achieve the same performance results.

7. Conclusion

This research provides a framework for automatically assigning the appropriate priority level for bug reports to avoid time-consuming and limited resources during the software testing process.

The proposed framework involves the use of text pre-processing methods (tokenization, stop words, and stemming) and then extracting important keywords from the description of the bug reports. A dataset was extracted from JIRA using the INTIX DWC company dashboard [4], which consists of five closed-source projects and containing more than 2000 bug reports. The dataset was divided into training and test cases and applied dataset variations (20% test and 80% training).

The proposed model has been validated on a dataset extracted from five real projects. The performance of the model is compared with two well-known ML algorithms, SVM and KNN. The results show that LSTM predicts and assigns the priority of the bug more accurately and effectively. LSTM significantly improves the average

F-measure in comparison to the other classifiers. The study showed that LSTM reported the best performance results based on all performance measures ($Accuracy = 0.908$, $AUC = 0.95$, F-measure $= 0.892$). This answers our research question, which suggests that LSTM outperforms the alternatives and improves performance.

In the future, we will validate other deep learning approaches on open-source projects like Eclipse and Mozilla. This includes experiments to evaluate the performance of different classifiers such as *Naive Bayes*, *RBF Networks*, and *Functional Trees*. Also, a future work direction might involve integrating the proposed framework with JIRA software.

References

[1] H. Rocha, G. De Oliveira, H. Marques-Neto, and M.T. Valente, "NextBug: a Bugzilla extension for recommending similar bugs," *Journal of Software Engineering Research and Development*, Vol. 3, No. 1, 2015, p. 3.

[2] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated prediction of bug report priority using multi-factor analysis," *Empirical Software Engineering*, Vol. 20, No. 5, 2015, pp. 1354–1383.

[3] J. Anvik, L. Hiew, and G.C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 361–370.

[4] S. Wang, W. Zhang, and Q. Wang, "FixerCache: Unsupervised caching active developers for diverse bug triage," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014, pp. 1–10.

[5] S. Mani, A. Sankaran, and R. Aralikatte, "DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging," *arXiv:1801.01275 [cs]*, Jan. 2018. [Online]. http://arxiv.org/abs/1801.01275

[6] M. Ohira, Y. Kashiwa, Y. Yamatani, H. Yoshiyuki, Y. Maeda, N. Limsettho, K. Fujino, H. Hata, A. Ihara, and K. Matsumoto, "A dataset of high impact bugs: Manually-classified issue reports," in *IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 518–521.

[7] Y. Tian, D. Lo, and C. Sun, "DRONE: Predicting Priority of Reported Bugs by Multi-factor Analysis.(2013)," in *29th IEEE International Conference on Software Maintenance (ICSM)*, 2013, pp. 22–28.

[8] Q. Umer, H. Liu, and Y. Sultan, "Emotion based automated priority prediction for bug reports," *IEEE Access*, Vol. 6, 2018, pp. 35 743–35 752.

[9] M. Mihaylov and M. Roper, *Predicting the Resolution Time and Priority of Bug Reports: A Deep Learning Approach*, Ph.D. dissertation, Department of Computer and Information Sciences, University of Strathclyde, 2019. [Online]. https://local.cis.strath.ac.uk/wp/extras/msctheses/papers/strath_cis_publication_2727.pdf

[10] P.A. Choudhary, "Neural network based bug priority prediction model using text classification techniques," *International Journal of Advanced Research in Computer Science*, Vol. 8, No. 5, 2017.

[11] L. Yu, W.T. Tsai, W. Zhao, and F. Wu, "Predicting defect priority based on neural networks," in *International Conference on Advanced Data Mining and Applications*. Springer, 2010, pp. 356–367.

[12] J. Kanwal and O. Maqbool, "Bug prioritization to facilitate bug report triage," *Journal of Computer Science and Technology*, Vol. 27, No. 2, 2012, pp. 397–412.

[13] M. Sharma, P. Bedi, K.K. Chaturvedi, and V.B. Singh, "Predicting the priority of a reported bug using machine learning techniques and cross project validation," in *12th International Conference on Intelligent Systems Design and Applications (ISDA)*. IEEE, 2012, pp. 539–545.

[14] M. Alenezi and S. Banitaan, "Bug reports prioritization: Which features and classifier to use?" in *12th International Conference on Machine Learning and Applications*, Vol. 2. IEEE, 2013, pp. 112–116.

[15] H.M. Tran, S.T. Le, S. Van Nguyen, and P.T. Ho, "An analysis of software bug reports using machine learning techniques," *SN Computer Science*, Vol. 1, No. 1, 2020, p. 4.

[16] V. Lyubinets, T. Boiko, and D. Nicholas, "Automated labeling of bugs and tickets using attention-based mechanisms in recurrent neural networks," in *IEEE Second International Conference on Data Stream Mining and Processing (DSMP)*. IEEE, 2018, pp. 271–275.

[17] G. Fan, X. Diao, H. Yu, K. Yang, and L. Chen, "Software defect prediction via attention-based recurrent neural network," *Scientific Programming*, Vol. 2019, 2019.

[18] 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, 2019, pp. 46 846–46 857.

[19] Z. Lin, F. Shu, Y. Yang, C. Hu, and Q. Wang, "An empirical study on bug assignment automation using Chinese bug data," in *3rd International Symposium on Empirical Software Engineering and Measurement.* IEEE, 2009, pp. 451–455.

[20] *Martix.* [Online]. https://www.martix.me/ (Last accessed May 17, 2020).

[21] *Hashfood.* [Online]. https://itunes.apple.com/us/app/hashfood/id1117103333?l=ar&ls=1&mt=8 (Last accessed May 17, 2020).

[22] *Tazej.* [Online]. https://itunes.apple.com/jo/app/%D8%B7%D8%A%D8%B2%D8%AC/id1150041871?mt=8 (Last accessed May 17, 2020).

[23] *Workspaces.* [Online]. https://itunes.apple.com/us/app/theworkspacesid1246555146?l=ar&ls=1&mt=8 (Last accessed May 17, 2020).

[24] *Maharah.* [Online]. https://play.google.com/store/apps/details?id=com.mharah.app&hl=ar (Last accessed May 17, 2020).

[25] *INTIX DWC Company.* [Online]. http://intix.net/ (Last accessed May 17, 2020).

[26] S.N. Ahsan, J. Ferzund, and F. Wotawa, "Program file bug fix effort estimation using machine learning methods for open source software Projects," *Institute for Software Technologist Technical*, 2009.

[27] L. Marks, Y. Zou, and A.E. Hassan, "Studying the fix-time for bugs in large open source projects," in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, 2011, pp. 1–8.

[28] P. Kaur and C. Singh, "A systematic approach for bug severity classification using machine learning's text mining techniques," *Journal of Computer Science and Information Technology*, Vol. 5, No. 7, 2016.

[29] *Bugzilla.* [Online]. https://www.bugzilla.org/ (Last accessed May 17, 2020).

[30] M. Günel, *Keras: Deep Learning library for Theano and TensorFlow.* [Online]. https://web.cs.hacettepe.edu.tr/~aykut/classes/spring2016/bil722/tutorials/keras.pdf (Last accessed May 17, 2020).

[31] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*, 1st ed. Beijing; Cambridge Mass.: O'Reilly Media, Jul. 2009.

[32] *Pycharm, The Python IDE for Professionals.* [Online]. https://www.jetbrains.com/pycharm/ (Last accessed May 17, 2020).

[33] Z. Imran, *Predicting bug severity in open-source software systems using scalable machine learning techniques*, mathesis, Youngstown State University, 2016.

[34] I. Mani and I. Zhang, "kNN approach to unbalanced data distributions: A case study involving information extraction," in *Proceedings of Workshop on Learning from Imbalanced Datasets*, Vol. 126, 2003.

[35] S. Hochreiter and J. Schmidhuber, "Long short--term memory," *Neural Computation*, Vol. 9, No. 8, 1997, pp. 1735–1780.

[36] M.N. Karim and S.L. Rivera, "Comparison of feed-forward and recurrent neural networks for bioprocess state estimation," *Computers and Chemical Engineering*, Vol. 16, 1992, pp. S369–S377.

[37] R. Santos, M. Rupp, S. Bonzi, and A.M. Fileti, "Comparison between multilayer feedforward neural networks and a radial basis function network to detect and locate leaks in pipelines transporting gas," *Chemical Engineering Transactions*, Vol. 32, 2013, pp. 1375–1380.

[38] L. Mei, R. Hu, H. Cao, Y. Liu, Z. Han, F. Li, and J. Li, "Realtime mobile bandwidth prediction using lstm neural network," in *International Conference on Passive and Active Network Measurement.* Springer, 2019, pp. 34–47.

[39] S. Ray, "Suppor vector machine algorithm in machine learning," Sep. 2017. [Online]. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

[40] W. Harrad, "A top machine learning algorithm explained: Support vector machines (svms)," Feb. 2020. [Online]. https://www.vebuso.com/2020/02/a-top-machine-learning-algorithm-explained-support-vector-machines-svms/

[41] M. Waseem, "Support vector machine in python," Nov. 2019. [Online]. https://www.edureka.co/blog/support-vector-machine-in-python/

[42] *sklearn.metrics.accuracy_score.* [Online]. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html (Last accessed April 30, 2020).

[43] U. Malik, *Implementing SVM and Kernel SVM with Python's Scikit-Learn.* [Online]. https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/ (Last accessed April 30, 2020).

[44] O. Harrison, *Machine Learning Basics with the K-Nearest Neighbors Algorithm*, 2018. [Online]. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761 (Last accessed April 30, 2020).

[45] *KNN Algorithm - Finding Nearest Neighbors.* [Online]. https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_knn_algorithm_finding_nearest_neighbors.htm (Last accessed April 30, 2020).

[46] Google Brain Team, *tensorflow. Develop and train ML models*, 2015. [Online]. https://www.tensorflow.org/ (Last accessed December 15, 2019).

[47] T. Mester, *Pandas Basics (Reading Data Files, DataFrames, Data Selection)*, 2019. [Online]. https://data36.com/pandas-tutorial-1-basics-reading-data-files-dataframes-data-selection/ (Last accessed May 17, 2020).

[48] D.P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[49] *train_test_split.* [Online]. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (Last accessed May 17, 2020).

[50] J. Brownlee, *How to Diagnose Overfitting and Underfitting of LSTM Models*, 2017. [Online]. https://machinelearningmastery.com/diagnose-overfitting-underfitting-lstm-models/ (Last accessed May 17, 2020).

[51] *PyQt5 Reference Guide.* [Online]. https://www.riverbankcomputing.com/static/Docs/PyQt5/ (Last accessed September 05, 2020).

[52] G. Yang, S. Baek, J.W. Lee, and B. Lee, "Analyzing emotion words to predict severity of software bugs: A case study of open source projects," in *Proceedings of the Symposium on Applied Computing*, 2017, pp. 1280–1287.

[53] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Computational Intelligence Magazine*, Vol. 13, No. 3, 2018, pp. 55–75.