Automatic Bug Priority Prediction using LSTM and ANN Approaches during Software Development

D.N.A. Dissanayake[1], R.A.H.M. Rupasingha[2#], and B.T.G.S. Kumara[3]
[1,2]Department of Economics and Statistics, Sabaragamuwa University of Sri Lanka, Belihuloya, Sri Lanka
[3]Department of Computing and Information Systems, Sabaragamuwa University of Sri Lanka, Belihuloya, Sri Lanka

[#]hmrupasingha@gmail.com

ABSTRACT The process of manually assign a priority value to a bug report takes time. There is a high chance that a developer may allocate the wrong value, and this can affect several important software development processes. To address this problem, the objective of this research incorporates three unique feature extraction approaches to create a model for automatically predicting the priority of bugs using the Long Short-Term Memory (LSTM) deep learning algorithm and Artificial Neural Network (ANN) algorithm. First, we collected approximately 20,500 bug reports from the Bugzilla; bug tracking system. Followed preprocessing, created models using two classifiers and feature vectors including Global Vectors for Word Representation (GloVe), Term Frequency-Inverse Document Frequency (TF-IDF), and Word2Vec used individually. The final classification results were determined by comparing the all results of the different models, which were integrated into an ensemble model. For evaluating the models, accuracy, recall, precision, and f-measure were used. The ensemble model produced the highest accuracy of 92% than other models as ANN model's accuracy was 80.28%, LSTM GloVe model's accuracy was 89.58%, LSTM TF-IDF model's accuracy was 88.94%, LSTM W2V model's accuracy was 84.84%. And also, higher recall, precision, and f-measure results were found in the ensemble model. Using the proposed model by LSTM-based ensemble approach we could automatically find the bug priority level of bug reports efficiently and effectively. In the future studies, intend to gather data from sources other than Bugzilla, such as JIRA or a GitHub repository. Additionally, try to apply other deep algorithms to improve the accuracy.

INDEX TERMS Bug Priority Prediction, Ensemble Model, LSTM

I. INTRODUCTION

A crucial step in the software development process is software maintenance which stands for changing, tweaking, and updating software and its features to produce a better version of it [1]. Developers and other responsible parties maintain software for a variety of purposes, including enhancing general software performance and fixing bugs after the software is released.

A bug repository is one of the most crucial software repositories and the most significant database in the software development process [2]. For updating and keeping information about problems that emerge or suggestions for improving the project, many software projects establish and maintain bug repositories. The people generate, store, update, and research every software defect in the software repositories. As a result, developers have to continuously update and produce different bug reports to aid in the creation and maintenance of software.

The most crucial task of the software that is being improved is bug fixing. To improve software systems, developers and project managers collect bug reports and look at Bug Tracking Systems (BTS) [3], sometimes referred to as issue tracking systems, such as JIRA [4] and Bugzilla [5], which assist developers in handling bug triaging and bug reporting [6].

The performance and quality of software systems may decline as a result of the numerous defects that exist in them. It is impossible to produce error-free software and many projects will be delivered with flaws because bugs are a regular occurrence [7]. Software creators enable users to submit defects in the BTS to enhance the upcoming version of the program. The following pre-defined fields are included in a bug report: the bug ID, content ID, title, error description, owner/author, status, priority, version, and severity [8]. The urgency of a defect's remedy is determined by bug priority.

Assigning a bug priority or bug prioritization is a very important task due to several reasons [9]. It facilitates a deeper comprehension of the bug and identifies potential solutions. After finding the bug, we can improve the program architecture to prevent it from becoming a greater issue. The bug that is creating the most issues is determined to have the highest priority. The priority of the bug determines the sequence in which the developer or project manager should fix it. With P1 denoting the highest priority and P5 denoting the lowest priority, a bug report's priority is assigned on a scale of P1 to P5. Bug prioritization is a manual process that requires a lot of

time because there are so many bug reports. When a defect is submitted, a developer looks into it and manually assigns priority to the pertinent bugs. The term "bug triaging" refers to this time-consuming manual process carried out by humans [10]. As a result, the likelihood of improper bug prioritizing is considerable. There may be a high possibility of incorrect bug prioritization as well. Automating the process of prioritizing bug reports is crucial for avoiding this serious problem. In this study, we suggested to build a model as a solution to the issue of identifying bugs with the highest priority.

This study's primary goal is to develop a model for automatically predicting the prioritization of bugs using ANN algorithm and LSTM deep learning algorithm by combining three feature extraction methods as a solution for above mentioned problem.

A bug priority prediction model can be useful in several ways. The machine learning and deep learning classifiers used for classifying the text of the bug reports when it comes to prioritizing bugs. After collecting data, they should pre-process. Then feature extraction is carried out utilizing three various techniques including TF-IDF, Word2Vec, GloVe with the LSTM algorithm and TF-IDF with ANN as algorithm. Three LSTM results were combined into ensemble model to take the final classification results with the comparison of individual model results of ANN and LSTM Models. Accuracy, recall, precision, and f-measure were used for measure the evaluation of the models.

The following is a summary of expected contributions of this paper.

I. Ensemble approach based on LSTM algorithm is proposed to automatic priority prediction of bug reports into five priority levels namely P1, P2, P3, P4 and P5.
II. The suggested strategy is based on analysing bug reports. The proposed methodology for bug priority prediction provides correct automatic priority levels for analysing and improving software systems on time.
III. LSTM three individual models, LSTM ensemble approach and ANN model are compared with each other to evaluate the performance of the proposed approach.

This paper is organized as follows. In Section 2, review existing literature. Section 3 explains the proposed methodologies. Research finding and evaluation of the results shown in Section 4. Finally, in Section 5 concludes the paper and discusses the recommendation.

II. LITERATURE REVIEW

A. Related Work

To identify the uniqueness of our research, it is important to review the existing literature in knowledge. The majority of current studies have used deep neural techniques, and relied on machine learning algorithms to forecast the priority levels in a bug report. We perform a critical analysis of the preceding works to show the originality of our study.

There were basically two main paths in early studies under the topic of bug report such as priority prediction and severity prediction [11]. Priority prediction of bug reports was recently carried out [12], using a CNN-based technique. Utilizing Natural Language Processing (NLP) techniques, done preprocessing on data bug descriptions, and created a classification model utilizing TCN, CNN, and SVM algorithms. Accuracy, Recall, Precision, and F1-score were used to evaluate how well the generated models performed. This study used a deep neural network-based algorithm, NLP techniques, and feature extractions to anticipate the priority levels of bug reports. And research findings state that CNN is best for priority prediction according to their study.

In order to eliminate manual bug prioritizing in [10], a software engineering domain repository was utilized to train and calculate the emotion value using emotion analysis. Based on input data, the CNN classifier makes a priority suggestion. The priority suggestions for the reports gathered from the Bugzilla and Eclipse projects were made using the CNN prioritization method. On average, proposed approach improves the F1 value by more than 6%. As well, some researchers Qasim Umer et al. [13], propose an automated approach for bug prediction of each issue report obtained using Eclipse data from the Bugzilla database. This method is based on emotion words. They coupled NLP techniques with machine learning algorithms like SVM, Naive Bayes classifier (NB), and Linear Regression (LR) to overcome the issue. As we select the LSTM approach for our study, Hani Bani Salameh et al. [14] constructed a deep learning RNN- LSTM network with five layers and compared the results with SVM and KNN for issue prediction based on more than 2000 JIRA bug reports. Results indicate that for performance-based accuracy, AUC, and f-measure, LSTM scored best. As in values, accuracy was 0.908, AUC was 0.95, and F measure was 0.892.

When it comes to severity prediction, the aim of previous studies is to investigate automated severity prediction because manual prioritization is time-consuming and tedious. The study [15], used NLP as a preprocessing strategy after extracting information from open-source project data and is based on the textual description that is under a deep neural network. Deep learning techniques such as CNN, LSTM, RF, and MNB were used for training and prediction, with CNN having the highest accuracy of all techniques. On average, it improves the F- score by 7.90% according to the results of the study. Additionally, severity prediction on data gathered through Bugzilla in [16], was carried out using the Bagging ensemble approach and the C4.5 classifier. The outcomes of comparing the two approaches indicate that the C4.5 classifier performs better at predicting severity of issue reports for cross component context and closed source software. According to the results J48 classifier

gain 79.82% of accuracy while bagging classification algorithm become highest accuracy among them while representing 81.27% accuracy.
The approach [17], organizes Mozilla and Eclipse bug reports into severity categories based on topics, then extracts features from each topic. Then, by assimilating traits from the LSTM and CNN algorithms, forecast the severity. In order to estimate the severity, they feed the CNN with extracted features as its input, and it uses its output to feed the LSTM. The performance of the suggested model was assessed by comparing it to the baseline in order to make better predictions.

Instead of single priority and severity prediction, there were some areas of researches under hybrid approach in both priority prediction and severity prediction. According to [18], they build a hybrid model for predict the defective areas of source code named CBIL. First using source code, they extracted the Abstract Syntax Tree (AST) tokens as vectors. Then CNN extracted the semantics of AST tokens. After that Bi- LSTM track the key vectors and reject other features to improve the accuracy of the model. Used dataset were seven open-source Java projects. According to their results, RNN accomplished the top performance. Not only that, but also in [19], they proposed a hybrid model for software defect prediction which combined SVM and RBF with MRMR feature selection. According to their results, MRMR gives better performance compared to SVM. According to other researchers Tanujit and Ashis proposed a novel hybrid methodology in their study [20], for improvement of defect prediction for software. In their study, they prove the theoretical consistency of their proposed model under more than ten NASA SDP datasets while showing the superiority of their proposed method.

As well as priority prediction there were some researchers done severity prediction under hybrid approach such as [21]. In their study thy proposed an approach for severity prediction based on the feature selection algorithm of the severity of each topic of data from Eclipse and Mozilla open-source projects. In the process they conducted, first classify issue reports by topic-based severity and extracted features from it. Then severity was predicted by learning characteristics from the LSTM and CNN algorithms. The comparison of summary of reviewed papers is shown in Table 1.

Table 1. Summary of existing studies

Ref. No	Data	Methodology	Objective/s	Limitations	Overcome limitations
[7]	JIRA	LSTM, SVM, KNN	Provides a framework for automate predict priority	2000 of small dataset	Use more than 20500 of data
[14]	Eclipse project & Bugzilla	CNN	To end manual prioritization of bug reports	Limited only one feature extraction	Apply three feature extraction methods
[12]	4 open-source projects	CNN	Predict the bug report's priority automatically	Limited only one feature extraction	
[13]	Bugzilla	Emotional Analysis, SVM	By avoiding manual Prioritization, predict priority that help developers to focus bugs resolution	Only consider emotional analysis	Considering bug prioritization using deep learning algorithms
[15]	Bugzilla	CNN, LSTM, RF, MNB	Automate prioritizing the severity	Only consider severity prediction	Based on bug priority prediction of bug reports
[16]	Bugzilla	J48 algorithm, Bagging algorithm	Find a new technique to avoid assigning incorrect severity using bagging ensemble method	Small dataset Only consider severity prediction	Increase data set into 20,500 & Focus on bug priority prediction
[17]	Eclipse and Mozilla projects	CNN, LSTM	Use topic-based feature selection and CNN-LSTM to predict severity	Only consider severity prediction	Based on bug priority prediction of bug reports
[18]	open-source Java projects	CNN & Bi-LSTM	Extract the semantics of source code for software defect prediction	Limited only one feature extraction	Apply three feature extraction methods
[19]	NASA Metrics Data Program	SVM & RBF	Build a model for effective to deal with the imbalance datasets in software defect prediction	Limited only one feature extraction	
[20]	NASA SDP dataset	-	Proves the theoretical consistency of Hellinger net	Provide only theoretical base	Use algorithms to evaluate each algorithm performance, use statistical measurements to evaluate models' effectiveness

[22]	Eclipse, Mozilla, Apache, and NetBeans datasets	SMOTE & IFSM	Predict the severity and priority of software bugs using the IFSM	Small dataset	Increase data set into 20,500
[23]	Mozilla, Eclipse, NetBeans, GCC	CNN, SVM, Random Forests & Logistic Regression, HAN	Build a Hierarchical Attention Network (HAN) model for prioritizing software bug reports	Use GloVe word embeddings only	Apply three feature extraction methods
[24]	JIRA deployed by Apache	Empirical study	Empirical study to explore the phenomenon of bug priority changes	Conduct only empirical study	Conduct a full approach using algorithms

B. Research Gap

The majority of existing researchers built a model for bug prioritization using just one feature vector. Some research used two feature extraction methods for result comparison and found a better one. There were no studies conduct combining more than two feature vectors. Also, there were no any studies which were combining unique feature extraction techniques with LSTM and ANN for bug priority prediction.

To cover this gap in the existing literature, we were able to apply a variety of three feature extraction techniques of TF-IDF, GloVe, and Word2Vec with an ensemble approach based on the LSTM deep learning algorithm. Then the result was compared with the individual results including the ANN algorithm result.

III. METHODOLOGY

This study automated the priority levels determined by bug reports using an algorithmic technique. During the scope of this research, a further four-step process of inquiry has been carried out. The first and second processes were collecting bug reports and pre-processing the data. In third step, features were retrieved from the pre-processed data using various extraction methods, such as TF-IDF, GloVe, and Word2Vec. As a fourth step, determine the bug reports related to their priority level using the LSTM and ANN algorithms by combining the three results taken from the three feature vector generations.

The research approach is described in detail in the below Figure 1.

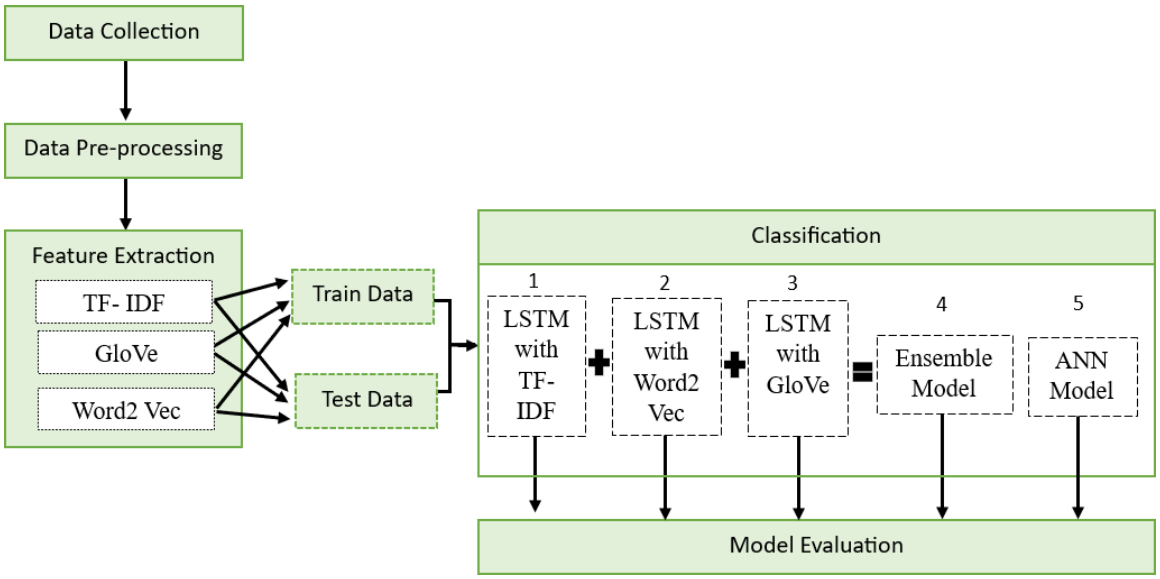


Figure 1. Research approach

A. Data Collection & Labeling

Software flaws also referred to as bugs, are found and tracked during software testing using bug tracking. Tracking issues or tracking defects can mention as other names for this process. There are many options for issue-tracking software, including Jira, Mantis, Redmine, Bugzilla, Backlog, and Bugnet.

Software bug reports are submitted to bug-tracking systems every day. Through Bugzilla, we collected over 21,000 bug reports under four open-source applications, including Firefox, Eclipse, Netbeans, and Open Office. The total bug reports we collected are shown in following Table 2.

Table 2. Total Bug Reports

Project	No of bug reports
Mozilla	4165
Eclipse	8478
Netbeans	4305
Open Office	4553
Total	21,501

The source of the bug reports was Bugzilla Bug tracking system. The bug ID (bugID), description (sd), classification (cl), product (pd), component (co), platform (rp), operating system (os), bug status (bs), resolution (rs), priority (pr), and severity (bsr) are the 11 columns in the CSV formatted data files that were created from the collected data. As a major feature, we select description (sd), which establishes the priority level. The remainder of the columns were removed. According to the data, description is the main variable which is affecting the bug priority level. While description act as an independent variable, priority level considers as a dependent target variable.

The data in Table 3 below show the 10 selected examples for the priority values P1, P2, P3, and P4, respectively.

Table 3. Data Examples

	Description	Priority
1	Scroll Up Down using Ctrl UP DOWN stopped to work	P1
2	Unable to create a new project	
3	Unable to update existing plugins and unable to install new plugins	P2
4	Can't display thumbnail images for plugins added since	
5	Copy and paste for an aux file do not work	P3
6	UI freezes in Plugins View	
7	Dead Link from Plugins window	P4
8	Error when uninstall IDE Win	
9	Allow only certain pages to modify fonts	P5
10	Intermittent session restores many windows timeout	

B. Data Pre-processing

The crucial phase in the machine learning process is data pre-processing [25]. The bulk of bug reports contains extraneous and meaningless details. For the classification model to produce better results, the data must be in the correct format. Pre-processing will improve the quality of the data set by eliminating those unnecessary data. It is the process of transforming unstructured data into a more understandable format. After handling the missing values, the text in the bug reports will be cleaned up using different pre-processing techniques as shown in the Figure 2.

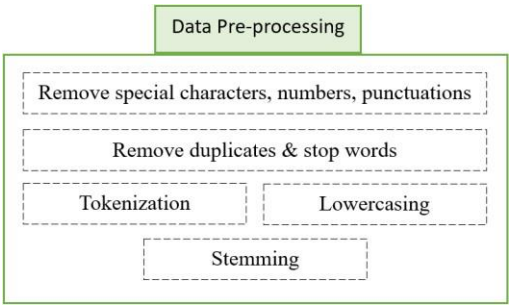


Figure 2. Steps of data pre-processing

Remove special characters, numbers, punctuations – Remove unnecessary special characters, and numbers, including punctuation from the data, and working with data that contains unnecessary special characters, numbers can be difficult. For this reason, we got rid of some punctuation, special letters, numbers (0–9), and symbols. (! " ” # $ % & \ ' ( ) * + , . / : ; < = > ? @ [\ \ ] ^ _ ` { | } ~).

Table 4. Examples for Remove special characters, numbers, punctuations

Description before remove special characters, numbers, punctuations	Description after remove special characters, numbers, punctuations
PDE quickfix creates invalid @Since tag	PDE quickfix creates invalid Since tag
OpenJ9: Git failing on master, builds blocked	OpenJ Git failing on master builds blocked
Use setUseHashlookup in internal org.eclipse.e4.ui.dialogs.filteredtree.FilteredTree	Use setUseHashlookup in internal org eclipse e ui dialogs filteredtree FilteredTree

Remove duplicates and stop words – Eliminate duplication and stop words since redundant data will result from duplicate data. Duplicate values in the data set were therefore eliminated. Moreover, often used stop words that lack precise definitions alone include "in," "the," "a," "our," "is," and "that." These stop-words will be eliminated during pre-processing.

Table 5. Examples for Remove duplicates & Stop words

Description before remove duplicates & stopwords	Description after remove duplicates & stopwords
PDE quickfix creates invalid Since tag	PDE quickfix creates invalid tag
OpenJ Git failing on master builds blocked	OpenJ Git failing master builds block
Use setUseHashlookup in internal org eclipse e ui dialogs filteredtree FilteredTree	Use setUseHashlookup internal org eclipse ui dialogs filteredtree FilteredTree

Tokenization – Tokenization is the process of breaking down the text into individual words, phrases, and clauses [26]. Unneeded symbols will be eliminated. To put it simply, tokenization eliminates all symbols from the text and divides it into tokens. By doing this, incoming data is divided into useful chunks that can be embedded in a vector space.

Table 6. Examples for Tokenization

Description before tokenization	Description after tokenization
PDE quickfix creates invalid tag	"['PDE', 'quickfix', 'creates', 'invalid', 'tag']"
OpenJ Git failing master builds block	"['OpenJ', 'Git', 'fail', 'master', 'builds', 'block']"
Use setUseHashlookup internal org eclipse ui dialogs filteredtree FilteredTree	"['Use', 'setUseHashlookup', 'internal', 'org', 'eclipse', 'ui', 'dialogs', 'filteredtree', 'FilteredTree']"

Lowercasing – Lowercasing refers to the process of changing all text and data to lowercase letters. Even though the terms "Home" and "home" have the same meaning, the vector space modeling recognizes them as two distinct words if they are not written in lowercase.

Table 7. Examples for Lowercasing

Description before lowercasing	Description after lowercasing
"['PDE', 'quickfix', 'creates', 'invalid', 'tag']"	"['pde', 'quickfix', 'creates', 'invalid', 'tag']"
"['OpenJ', 'Git', 'fail', 'master', 'builds', 'block']"	"['openJ', 'git', 'fail', 'master', 'builds', 'block']"
"['Use', 'setUseHashlookup', 'internal', 'org', 'eclipse', 'ui', 'dialogs', 'filteredtree', 'FilteredTree']"	"['use', 'setusehashlookup', 'internal', 'org', 'eclipse', 'ui', 'dialogs', 'filteredtree', 'filteredtree']"

Stemming – In the English language, a single sentence can take many different forms. These inconsistencies in a text cause data in machine learning models to become redundant. Due to this, it will be impossible to produce the desired results. Thus, stemmed terms in the data set ought to be eliminated. Words are being reduced to their stems in this process. As an illustration, the terms "take," "takes," "taken," and "took" can all be replaced with the one word "take."

Table 8. Examples for Stemming

Description before stemming	Description after stemming
"['pde', 'quickfix', 'creates', 'invalid', 'tag']"	"['pde', 'quickfix', 'create', 'invalid', 'tag']"
"['openJ', 'git', 'fail', 'master', 'builds', 'block']"	"['openJ', 'git', 'fail', 'master', 'build', 'block']"
"['use', 'setusehashlookup', 'internal', 'org', 'eclipse', 'ui', 'dialogs', 'filteredtree', 'filteredtree']"	"['use', 'setusehashlookup', 'internal', 'org', 'eclipse', 'ui', 'dialogs', 'filteredtree', 'filteredtree']"

After the data had been appropriately gathered and captured from the Bugzilla, we apply some data pre-processing techniques to clean the bug reports and eliminate the superfluous content. There were no longer any tags, URLs, links, or numbers. As a result, deleting them aids in shrinking the feature space.

C. Feature Extraction

After preprocessing, data must be converted into features for modeling. Raw text input data cannot be directly used to use machine learning techniques. The acquisition of contextual characteristics of the text is required in order to convert it into feature vectors. Symbolic and numeric characters are represented by features in machine learning and pattern recognition [27]. Three different feature vector approaches, TF-IDF, GloVe, and Word2Vec were utilized in this inquiry.

1) TF-IDF

Text input will simply be converted into a numerical format known as a vector form capable of machine learning techniques by using this feature vector. A statistical assessment called TF-IDF [28] looks at a word's relevance to each document in a set of documents. A word's TF-IDF is determined by multiplying two separate metrics.

Term Frequency (TF): The phrase "term frequency" refers to how frequently a word appears in a document. This is calculated by dividing the number of times a word appears in a document by the total number of words in the document. The calculation is shown in following (1).

$$TF = \frac{(\#\ of\ repetitionsof\ word\ in\ a\ document)}{(\#\ of\ words\ in\ a\ document)} \quad (1)$$

Inverse Document Frequency (IDF): The inverse document frequency measures how frequently a word appears in a group of documents. This indicates how uncommon a word is over the full corpus of documents. This value will be close to 0, if the word is widely used and frequently found in a document, else it will be 1. The calculation is shown in following (2).

$$IDF = \frac{(\#\ of\ documents)}{(\#\ of\ words\ in\ a\ document)} \quad (2)$$

TF-IDF: These two numbers are multiplied to provide the TF-IDF score of a word in a document. The word becomes more significant in that specific document the higher the score. The calculation is shown in following (3).

$$TF - IDF = TF * IDF \quad (3)$$

2) GloVe

The GloVe [29],an unsupervised learning method for Word Representation, is a popular algorithm for generating word embeddings in machine learning. Word embeddings are dense vector representations of words that capture semantic and syntactic relationships between words based on their contexts in a given corpus of text. The primary goal of GloVe is to create word embeddings that capture the meaning of words by leveraging the statistics of word co-occurrence in a large corpus. It combines elements from two different approaches to word embeddings: count-based methods like Latent Semantic Analysis and predictive methods like Word2Vec.

3) Word2Vec

Word2Vec [30] is a popular algorithm in machine learning that is used for representing words as numerical vectors in a high-dimensional space. It was introduced by Tomas Mikolov et al. at Google in 2013 and has since become a fundamental tool for NLP tasks. The main idea behind Word2Vec is to capture the semantic and syntactic relationships between words by learning vector representations based on their contextual usage in a given corpus of text. The algorithm operates on the assumption that words appearing in similar contexts tend to have similar meanings. For example, in the sentence "The cat sat on the mat," the words "cat" and "mat" are more likely to be similar in meaning because they both appear in the context of a sentence about sitting on an object.

D. Classification

Then LSTM deep learning algorithm and ANN algorithms are used to forecast the priority levels in a bug report. Cleaning and feature extraction of the data resulted in the separation of the data into training and testing data, with training data accounting for 70% and testing data for the remaining 30% by the experiment. The different approaches generate for priority prediction by loading it into ANN and LSTM independently.

1) LSTM

LSTM is a type of recurrent neural network, (RNN [31]) architecture that is designed to handle and model long-term dependencies in sequential data. It overcomes the limitations of traditional RNNs by introducing memory cells and gating mechanisms. The key idea behind LSTM is the concept of a memory cell, which allows the network to remember information over long sequences. The memory cell is responsible for storing and updating information, selectively forgetting or retaining it based on its relevance to the current context. Overall, LSTM is a powerful RNN variant that enables the modeling of long-term dependencies and has been widely adopted in various domains of machine learning and natural language processing due to its ability to effectively handle sequential data.

2) ANN

Artificial neural networks, often known as neural networks, are models that use computer techniques to imitate the behavior of neuron-based biological systems [32]. ANN is with machine learning and pattern recognition capabilities. The central nervous system of animals serves as the model's inspiration. This network of "neurons" may compute values from input data. Three layers, including an input layer, a hidden layer, and an output layer, may be present in a neural network.

LSTM deep learning algorithm and the ANN algorithm were chosen due to their strengths in handling sequential data, learning complex relationships [33], and their established effectiveness in similar machine learning tasks based on the literature review.

In this study, TF-IDF, GloVe, and Word2Vec feature vectors were used to create three LSTM prediction model separately. Then, three distinct models that were in alignment with the LSTM algorithm were combined to make an ensemble model. When each model predicts the priority, we selected the majority by comparing three results generated by three models as below table 9 shows.

Table 9. Selecting majority value using three LSTM models

Description	GLOVE	TF-IDF	W2V	Majority	Ensemble Result
Performance loss in composite WM paint	1	1	1	1	1
Validator incorrectly flags conditionally declared class as a PHP error	2	3	3	3	3
Crash when opening a presentation macOS	4	4	5	4	4

Then assigning those majority value as the ensemble model result, we compared it with the individual algorithm results including ANN model to select the best one. The process of ensemble model creation is explained in Figure 3.

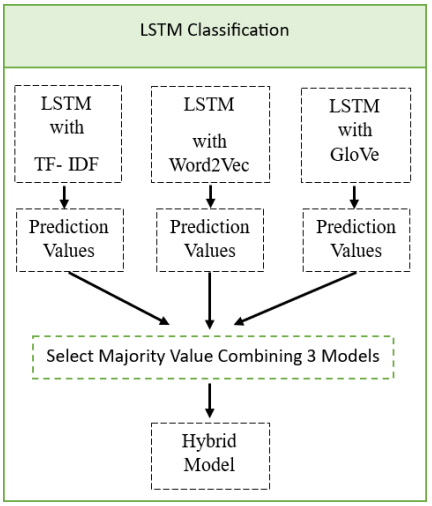


Figure 3. Process of making ensemble model

In parallel, ANN model also created using same three feature vectors and we were identified results were lower than the LSTM Model results. And TF-IDF is the only method which is generating highest accuracy among those three feature vector generations with ANN. So, we decided to use ANN with TF-IDF for further comparison.

After creation of all five models, considering actual priority values and predicted priority values under all models including ANN model, LSTM GloVe, LSTM Word2Vec, LSTM TF-IDF, and ensemble model we done the validation to find a better model using selected 5000 total data set.

IV. RESULTS & EVALUATION

The experimental platform used Microsoft Windows 11 on a PC with processor 8th Gen Intel(R) Core (TM) i3-8145U CPU @ 2.10 GHz 2.30 GHz, 4.00 GB RAM to training, testing, and implementing model.

The evaluation is based on three feature vector extraction methods with LSTM algorithm and ANN algorithm under collected bug reports via Bugzilla. The experiment is designed to classify the descriptions of bug reports into five priority levels using above feature vectors and algorithm. The model is being developed in Python programming language.

Accuracy, precision, recall, and f-measure were computed for ANN and LSTM algorithm under three feature vectors using below (4) – (7) respectively.

Following (4) used to evaluate the accuracy of the model. To do that, total number of correct predictions should divide by total number of predictions. Following (5) used for measure the actually correct proportion of positive identifications. We used recall and f-measure for the evaluation and calculated using (6) and (7) respectively.

$$Accuracy = \frac{No\ of\ Correct\ Predictions}{Total\ Prediction} \quad (4)$$

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad (5)$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \quad (6)$$

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} \quad (7)$$

A. Results of Model Implementation

First, we implement the three LSTM models under different feature vectors and ANN model with TF-IDF. In this section shows the results based on above evaluation measurements on full dataset of 20,501 bug reports. Based on the performance of each model, Figure 4 shows the accuracy results for ANN and LSTM classifier under three feature extraction methods. According to the results, the accuracy of ANN model is 80%, LSTM TF-IDF model is 78.10%, LSTM GloVe model is 81.47% and LSTM Word2Vec model is 75%. Based on the results, LSTM GloVe model shows the highest accuracy among all four models.

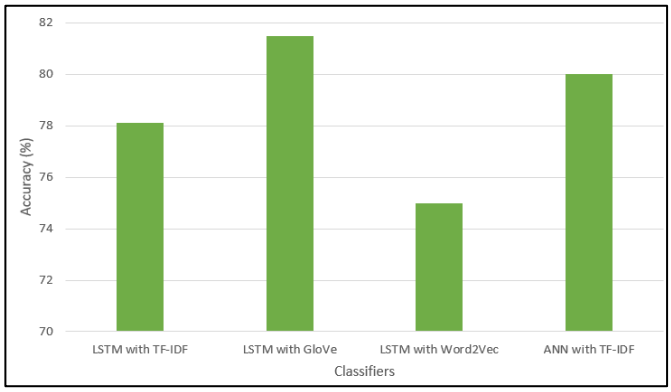


Figure 4. Model Implementation Accuracy of ANN and LSTM under TF-IDF, GloVe, Word2Vec

Table 9 shows the evaluation result for precision, recall, and f-measure of ANN model and LSTM model under three different feature vectors.

Table 10. Precision, recall, f-measure of ANN & LSTM under TF-IDF, GloVe, Word2Vec

Feature Vector	Precision	Recall	F- measure
LSTM TF-IDF	96%	81%	87%
LSTM GloVe	98%	83%	89%
LSTM Word2Vec	94%	77%	84%
ANN TF-IDF	93%	83%	83%

Furthermore, evaluation was consider using different percentages (57%, 67%, 77% and 87%) of training data. Accuracy values for ANN and LSTM under three feature vectors were as in Table 10. By considering the results of different training data sizes, we identified 77% of training data size as optimum value for our evaluation process. Then we divide dataset into 77% of training data and 33% of testing data.

Table 11. Evaluation of accuracy according to the different training data set

Classifier	Training data size			
	57%	67%	77%	87%
LSTM TF- IDF Accuracy (%)	78%	78%	78%	78%
LSTM GloVe Accuracy (%)	78%	77%	81%	78%
LSTM W2V Accuracy (%)	75%	74%	75%	74%
ANN TF-IDF Accuracy (%)	72%	76%	80%	78%

And also, hyperparameters of the ANN and LSTM classifiers were found, and the obtained optimal value for hyperparameters is provided in Table 11 as below and evaluation of accuracy based on those parameters shown in Figure 5 – Figure 8.

Table 12. ANN - LSTM Classifier Hyperparameters

Classifier	Optimum values discovered for the hyperparameters
LSTM with TF-IDF	Epochs= 100, batch_size= 52, optimizer = 'adam'
LSTM with GloVe	Epochs= 200, batch_size= 33, optimizer = 'adam'
LSTM with W2V	Epochs= 200, batch_size= 43, optimizer = 'adam'
ANN with TF-IDF	Epochs=250, batch_size=43, optimizer = 'adam'

B. Results of Data Validation

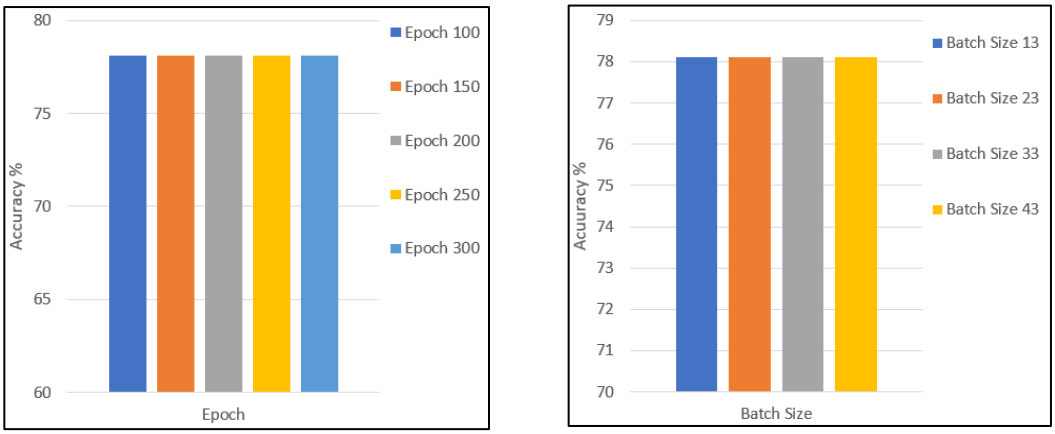
Priority values were predicted using LSTM three models which we implemented using TF-IDF, GloVe and Word2Vec and ANN with TF-IDF under validation dataset of 5000 bug reports. Then we check the algorithm outcomes by combining different feature extraction techniques results, we found majority priority value for each bug description. By considering that majority value we got the ensemble model priority value as shown in the Table 12 examples.

Table 13. Generating majority value for ensemble model

Bug Report	GloVe Priority	TF IDF Priority	Word2Vec Priority	Majority Value (LSTM Ensemble model)
1	P1	P1	P1	P1
2	P2	P1	P1	P1
3	P3	P3	P3	P3
4	P4	P3	P3	P3
5	P5	P5	P4	P5

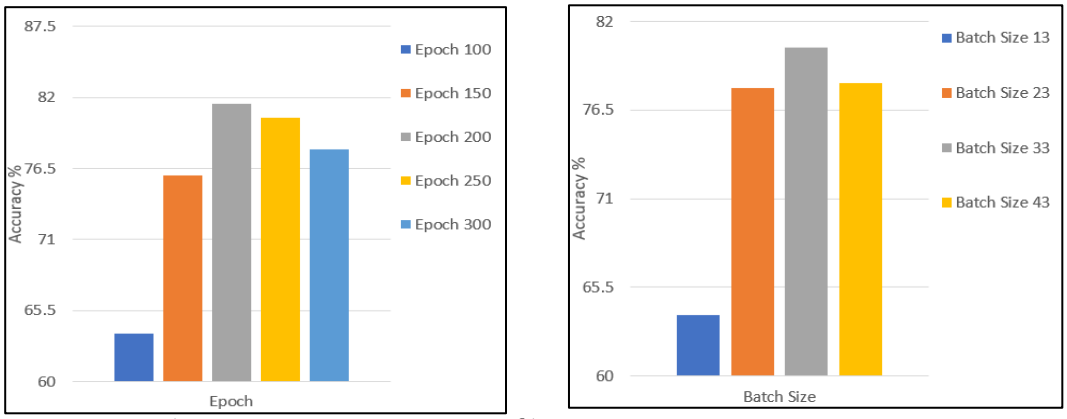
Then considering actual priority values and predicted priority values under five models including ANN model, LSTM GloVe, LSTM Word2Vec, LSTM TF-IDF, and ensemble model we done the evaluations to find a better model

.



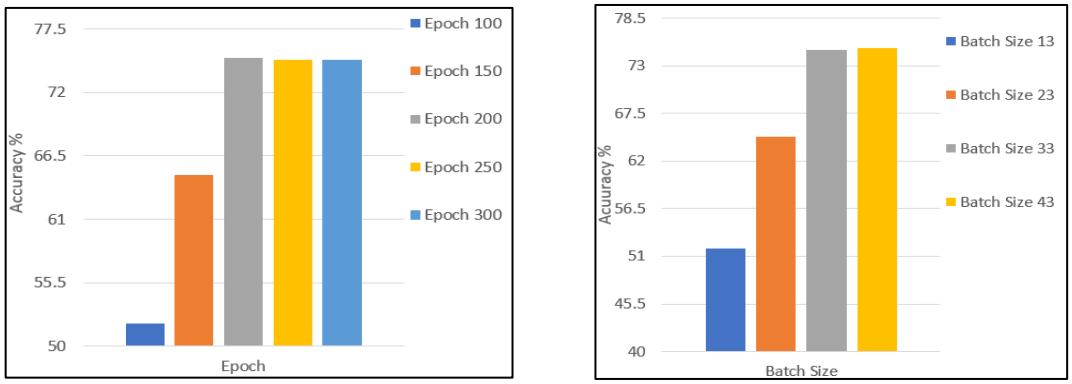
(a) (b)

Figure 5: (a) Epochs accuracy and (b) Batch sizes accuracy of LSTM with TF-IDF



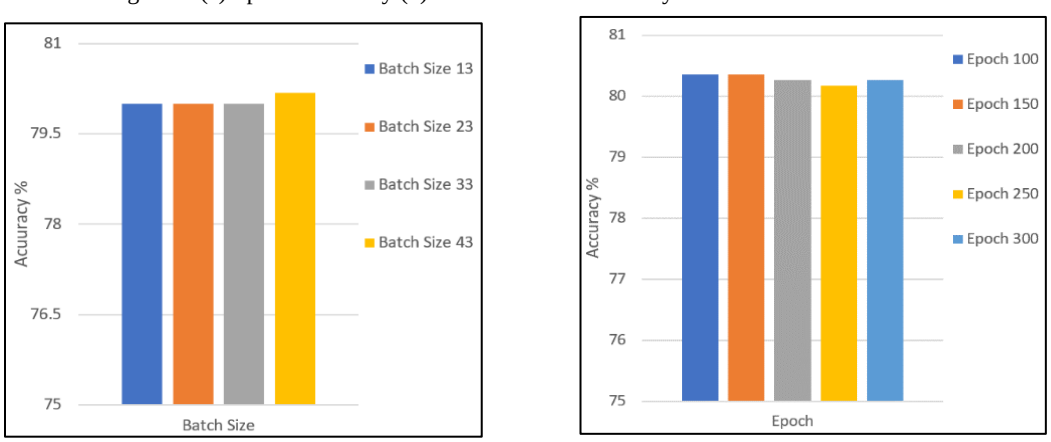
a) (b)

Figure 6: (a) Epochs accuracy (b) and Batch sizes accuracy of LSTM with GloVe



(a) (b)

Figure 7: (a) Epochs accuracy (b) and Batch sizes accuracy of LSTM with Word2Vec



(a) (b)

Figure 8: (a) Epochs accuracy (b) and Batch sizes accuracy of ANN with TF-IDF

When considering actual priority and predicted priority values ensemble model achieving more accuracy than other three individual models under ANN and LSTM classifier. Following Table 13 shows the summary of predicted values and its accuracy. Here total predictions are 5000.

Table 14. Summary of prediction and its accuracy with the ensemble model using validation data

Classifier	True Predictions	Wrong Predictions	Accuracy
ANN TF-IDF Model	4014	986	80%
LSTM GloVe Model	4479	521	89%
LSTM TF-IDF Model	4447	553	88%
LSTM W2V Model	4242	758	84%
LSTM Ensemble Model	4600	400	92%

Furthermore, following Recall, Precision, and F- measure were calculated under five prediction models based on validation data shown as Table 14.

Table 15. Precision, recall, f-measure of ANN, LSTM & Ensemble models using validation data

Models	Precision	Recall	F- measure
ANN TF-IDF Model	79%	81%	88%
LSTM TF-IDF Model	89%	95%	91%
LSTM GloVe Model	89%	96%	93%
LSTM Word2Vec Model	89%	88%	78%
LSTM Ensemble Model	94%	98%	95%

When comparing all the five models, ensemble model achieved the highest accuracy as well as best performance among all other evaluation metrics. It means ensemble model can gain more reliable and accurate predictions than other four models.

C. Results of statistical test (Student t-test)

It's important to use statistical hypothesis test to select the final model. Statistical significance tests are designed to address and compare the performance of machine learning models. In this study we used student t-test for find a better model. In the case of comparing the performance of models, we have to select two models to perform the paired Student's t-test. Among the above five models, based on the highest accuracy, LSTM GloVe model and Ensemble model were compared. In this student t-test which can compare the means of two independent samples to see if they are significantly different from each other. We conducted a two-sample t-tests to compare the means of these two sets of accuracies as below table.

Table 16. Results of t-test

Measurement	Value
t-statistic	6.5027
p-value	0.0001

The t-statistic is significantly high at 6.5027 while p-value is extremely low at 0.0001 Since the p-value is much less than 0.05, we reject the null hypothesis that there is no difference between the accuracies of two models. This indicates that the observed difference in accuracies between the LSTM GloVe model and the Ensemble model is statistically significant.

Therefore, based on this statistical evidence, we concluded that the proposed models perform statistically significant. Based on the accuracies, the ensemble model selected as the better model among these two algorithms.

IV. CONCLUSION

The process of manually assigning a bug priority value to a bug report takes time. There is a chance that a developer will reassign a wrong value, and this can affect several important software development processes. The main objective of this research is to incorporate three feature extraction approaches to create a model for automatically predicting the priority of bugs using the ANN and LSTM deep learning algorithm as a solution to the aforementioned problem.
We collect approximately 20,500 bug reports from Bugzilla; bug tracking system. Following preprocessing, created three models using the LSTM classifier and three unique feature vectors including GloVe, TF- IDF, and Word2Vec. After comparing the three LSTM models' outputs, the majority value was determined for creating an ensemble model. In parallelly, ANN model was built under TF-IDF feature vector. After validating those five models on 5000 bug reports and comparing outcomes, it was found that the ensemble model generated the most accurate results than other four models. In the prediction procedure, the ANN model's accuracy was 80.28%, LSTM GloVe model's accuracy was 89.58%, the LSTM TF-IDF model's accuracy was 88.94%, the LSTM W2V model's accuracy was 84.84%, and the accuracy of the ensemble model was 92%. For the purpose of evaluating the models, accuracy, recall, precision, and f-measure were used. Ensemble model achieving the highest values in all evaluation matrixes and shows it was the best method to predict the bug priority value in bug fixing during the software development process. As well as based on this statistical evidence, the models are statistically significant with higher t-statistic value 6.5027 and p-value 0,0001.

These research findings will assist programmers, software developers and project managers in fixing bugs in software systems more quickly than before. As well as new researchers can gain knowledge regarding automate the bug priority prediction. In the future studies, we intend to gather data from sources other than Bugzilla, such as JIRA or a GitHub repository. Additionally, we try to apply other deep learning algorithms to improve the accuracy. And also, we are planning to improve the statistical test using all the proposed models in the future.

REFERENCES

[1] K. Moran, "Enhancing android application bug reporting," *10th Joint Meeting Foundation*, no. Aug, 2015, pp. 1045 - 1047, 2015.

[2] J. Xuan, H. Jiang, Z. Ren and W. Zou, "Developer Prioritization in Bug Repositories," in *34th International Conference on Software Engineering (ICSE)*, 2012.

[3] "Debian-wiki portal," 08 04 2022. [Online]. Available: https://wiki.debian.org/BTS. [Accessed 25 June 2023].

[4] "Jira Issue Tracker," Atlassian, [Online]. Available: https://www.atlassian.com/software/jira/. [Accessed January 2023].

[5] "Bugzilla Issue Tracker," Mozilla, [Online]. Available: https://www.bugzilla.org/. [Accessed January 2023].

[6] X. Xia, D. Lo, X. Wang and B. Zhou, "Accurate developer recommendation for bug resolution," in *20th Working Conference Reversen Eng. (WCRE)*, Oct., 2013.

[7] H. Bani-Salameh, M. Sallam and B. Al shboul, "A Deep-Learning-Based Bug Priority Prediction Using RNN-LSTM Neural Networks," *e-Information Software Engineering Journal*, vol. 15, no. 1, pp. 29-45, 2021.

[8] "Bugzilla - Reporting a Bug," www.Bugzilla.org, June 2023. [Online]. Available: https://www.bugzilla.org/contributing/reporting_bugs.html. [Accessed 14 June 2023].

[9] R. Harris, "Singlemindconsulting," 14 August 2020. [Online]. Available: https://www.singlemindconsulting.com/blog/prioritize-bug-fixes-vs-product-features/. [Accessed 25 June 2023].

[10] H. L. a. I. I. Q. Umer, "CNN-based automatic prioritization of bug reports," *IEEE transactions on reliability*, Vols. 69, no. 4, pp. 1341-1354, 2020.

[11] "Browserstack - Bug Severity Vs Priority," www.browserstack.com, 2023. [Online]. Available: https://www.browserstack.com/guide/bug-severity-vs-priority\. [Accessed 14 June 2023].

[12] R. Rathnayake, B. Kumara and E. Ekanayake, "CNN-Based Priority Prediction of Bug Reports," in *International Conference on Decision Aid Science and Application (DASA)*, 2021.

[13] Q. Umer, H. Liu and Y. Sultan, "Emotion Based Automated Priority Prediction for Bug Reports," *IEEE Access*, vol. 6, no. July 2, 2018, pp. 35743-35752, 2018.

[14] Q. Umer, H. Liu and . I. Illahi, "CNN-based automatic prioritization of bug reports," *IEEE trans. reliab*, vol. no. 4, no. 69, pp. 1341-1354, 2020.

[15] W. Y. Ramay, Q. Umer, C. Zhu, X. U. C. Yin and . I. Inam, "Deep Neural Network-Based Severity Prediction of Bug Reports," *IEEE Access*, vol. 7, no. 2019, pp. 46846 - 46857, 2019.

[16] P. Latha and M. Marlakunta, "Predicting the Severity of Bug Reports using Classification Algorithms," researchgate.net, Bangalore, India, 2021.

[17] J. KIM and G. YANG, "Bug Severity Prediction Algorithm Using Topic-Based Feature Selection and CNN- LSTM Algorithms," *IEEE Access*, vol. 10, no. 14, pp. 94643-94651, 2022.

[18] A. B. Farid, E. M. Fathy, A. S. Eldin and L. A. Abd-Elmegid, "Software defect prediction using hybrid model (CBIL) of convolutional neural network (CNN) and bidirectional long short-term memory (Bi-LSTM)," *PeerJ Computer Science*, p. 19, 2021.

[19] M. W. Thant and N. T. Aung, "Software Defect Prediction using Hybrid Approach," in *University of Information Technology, Yangon*, Myanmar.

[20] T. Chakraborty and A. K. Chakraborty, "Hellinger Net: A Hybrid Imbalance Learning Model to Improve Software Defect Prediction," Cornell University Library, 2020 September 12.

[21] G. Y. J. KIM, "Bug Severity Prediction Algorithm Using Topic-Based Feature Selection and CNN- LSTM Algorithms," *IEEE Access*, vol. 19, no. 14 September 2022, pp. 94643 - 94651, September 2022.

[22] R. R. Panda and N. K. Nagwani, "Software bug severity and priority prediction using SMOTE and Intuitionistic fuzzy similarity measure," *Applied Soft Computing*, vol. 150, 2024.

[23] A. Yadav and S. S. Rathore, "A Hierachical Attention Networks based Model for Bug Report Prioritization," in *17th Innovations in Software Engineering Conference (ISEC-2024)*, Bangalore, India, 2024.

[24] Z. Li, G. Cai, Q. Yu, P. Liang, R. Mo and H. Liu, "Bug priority change: An empirical study on Apache projects," *Journal of Systems and Software*, vol. 212, 2024.

[25] "Java Point - Data Preprocessing in Machine Learning," [Online]. Available: https://www.javatpoint.com/data-preprocessing-machine-learning. [Accessed 25 June 203].

[26] "Gartner," 2023. [Online]. Available: https://www.gartner.com/en/information-technology/glossary/tokenization#:~:text=Tokenization%20refers%20to%20a%20process,requires%20strong%20protections%20around%20it.. [Accessed May 2023].

[27] "ScienceDirect - Pattern Recognition," [Online]. Available: https://www.sciencedirect.com/journal/pattern-recognition. [Accessed 25 June 2023].

[28] A. Simha, "Capital One - Understanding TF-IDF for machine learning," 7 October 2021. [Online]. Available: https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/. [Accessed 25 June 2023].

[29] "Stanford University," [Online]. Available: https://nlp.stanford.edu/projects/glove/. [Accessed 25 June 2023].

[30] shrisikotaiah, "Geeksforgeek," [Online]. Available: https://www.geeksforgeeks.org/word-embeddings-in-nlp/#:~:text=Word%20Embedding%20or%20Word%20Vector,can%20represent%2050%20unique%20features..

[31] "IBM - RNN," [Online]. Available: https://www.ibm.com/topics/recurrent-neural-networks. [Accessed 25 June 2023].

[32] "Scholar- Google," [Online]. Available: https://scholar.google.com/scholar?q=neuron-based+biological+systems&hl=en&as_sdt=0&as_vis=1&oi=scholart. [Accessed 23 June 2023].

[33] P. Srivatsavya, "LSTM - Implementation, Advantages and Disadvantages," 5 October 2023. [Online]. Available: https://medium.com/@prudhviraju.srivatsavaya/lstm-implementation-advantages-and-diadvantages-914a96fa0acb#:~:text=Handling%20Long%20Sequences%3A%20LSTMs%20are,NLP)%20and%20time%20series%20analysis.. [Accessed 30 June 2024].

[34] M. S. a. B. A. s. H. Bani-Salameh, *e-Information Software Engineering*, vol. 15, p. no. 1, 2021.

[35] J. Kim and G. Yang, "Bug Severity Prediction Algorithm Using Topic-Based Feature Selection And CNN-LSTM Algorithm," *IEEE Access*, vol. 10, pp. 94643-94651, 2022.

[36] "java Point - Data Preprocessing in Machine Learning," [Online]. Available: https://www.javatpoint.com/data-preprocessing-machine-learning. [Accessed 25 June 2023].

Acknowledgment: There are no any special parties to acknowledge.
Financial interest: The authors declare they have no relevant financial or non-financial interests to disclose.
Conflict of interest: The authors have no conflicts of interest to declare.

AUTHOR BIOGRAPHIES

D. N. A. Dissanayake was born in June 1998. She graduated from Sabaragamuwa University of Sri Lanka, with a second upper class bachelor's degree in Information and Communication Technology. Her recent research interests include machine learning, artificial intelligence, the development of application based on data mining.

R.A.H.M. Rupasingha received her BSc in 2013 from Sabaragamuwa University in Sri Lanka. She obtained her MSc and PhD in 2016 and 2019, respectively, from the School of Computer Science and Engineering, the University of Aizu, Japan. Currently, she is a senior lecturer in Sabaragamuwa University in Sri Lanka. Her research interests include machine learning, ontology learning, data mining and recommendation

B. T. G. S. Kumara received the bachelor's degree in 2006 from Sabaragamuwa University of Sri Lanka. He received the master's degree in 2010 from University of Peradeniya, Sri Lanka and he received the PhD from School of Computer Science and Engineering, University of Aizu, Japan in 2015. Currently, he is a professor in Sabaragamuwa University in Sri Lanka. His research interests include semantic web, data mining, machine learning, web service discovery and composition.