Hindawi
Security and Communication Networks
Volume 2023, Article ID 9785401, 1 page
https://doi.org/10.1155/2023/9785401

Retraction

Retracted: An Automated Approach for the Prediction of the Severity Level of Bug Reports Using GPT-2

Security and Communication Networks

Received 5 December 2023; Accepted 5 December 2023; Published 6 December 2023

Copyright © 2023 Security and Communication Networks. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This article has been retracted by Hindawi, as publisher, following an investigation undertaken by the publisher [1]. This investigation has uncovered evidence of systematic manipulation of the publication and peer-review process. We cannot, therefore, vouch for the reliability or integrity of this article.

Please note that this notice is intended solely to alert readers that the peer-review process of this article has been compromised.

Wiley and Hindawi regret that the usual quality checks did not identify these issues before publication and have since put additional measures in place to safeguard research integrity.

We wish to credit our Research Integrity and Research Publishing teams and anonymous and named external researchers and research integrity experts for contributing to this investigation.

The corresponding author, as the representative of all authors, has been given the opportunity to register their agreement or disagreement to this retraction. We have kept a record of any response received.

References

[1] M. kamal, S. Ali, A. Nasir, A. Samad, S. Basser, and A. Irshad, "An Automated Approach for the Prediction of the Severity Level of Bug Reports Using GPT-2," *Security and Communication Networks*, vol. 2022, Article ID 2892401, 11 pages, 2022.

Hindawi
Security and Communication Networks
Volume 2022, Article ID 2892401, 11 pages
https://doi.org/10.1155/2022/2892401

WILEY | Hindawi

RETRACTED

Research Article

An Automated Approach for the Prediction of the Severity Level of Bug Reports Using GPT-2

Mohsin kamal,[1] **Sikandar Ali**,[2] **Anam Nasir**,[1] **Ali Samad**,[3] **Samad Basser**,[4] **and Azeem Irshad**[5]

[1]*Department of Computer Science, COMSATS University, Islamabad, Pakistan*
[2]*Department of Information Technology, The University of Haripur, Haripur 22620, Khyber Pakhtunkhwa, Pakistan*
[3]*Faculty of Computing, The Islamia University of Bahawalpur, Bahawalpur 63100, Pakistan*
[4]*Department of Computer System Engineering, University of Engineering and Technology, Peshawar 25000, Pakistan*
[5]*Department of Computer Science and Software Engineering, International Islamic University, Islamabad, Pakistan*

Correspondence should be addressed to Sikandar Ali; sikandar@uswat.edu.pk and Azeem Irshad; irshadazeem2@gmail.com

Received 8 January 2022; Revised 21 April 2022; Accepted 4 May 2022; Published 29 May 2022

Academic Editor: Muhammad Arif

Copyright © 2022 Mohsin kamal et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Manual investigation is warranted in traditional approaches for estimating the bug severity level, which adds to the effort and time required. For bug severity report prediction, numerous automated strategies have been proposed in addition to manual ones. However, the current bug report predictors by facing several issues, such as overfitting and weight computation, and therefore, their efficiency for specific levels of data noise needs to improve. As a result, a bug report predictor is required to solve these concerns (e.g., overfitting and avoiding weight calculation, which increases computing complexity) and perform better in the situation of data noise. We use GPT-2's features (limiting overfitting and supplying sequential predictors rather than weight computation) to develop a new approach for predicting the severity level of bug reports in this study. The proposed approach is divided into four stages. First, the bug reports are subjected to text preprocessing. Second, we assess each bug report's emotional score. Third, each report is presented in vector format. Finally, an emotion score is assigned to each bug report, and a vector of each bug report is produced and sent to GPT-2. We employ statistical indicators like recall, precision, and *F*1-score to evaluate the suggested method's effectiveness and efficacy. A comparison was also made using state-of-the-art bug report predictors such as Random Forest (RF), Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) Network, Support Vector Machine (SVM), XGBoost, and Naive Bayes Multinomial (NBM). The proposed method's promising result indicates its efficacy in bug information retrieval.

1. Introduction

As indicated by the examination local area, the quality and strength of the product advancement are related to the data extricated from the bug reports [1]. Regularly, bug reports are chosen in light of the seriousness level of bugs. Thus, the seriousness level characteristic of a bug report is the most essential element for getting the progression and upkeep sorted out, particularly in Free Open-Source Programming [2]. The seriousness level gauges the impact that a bug has on a useful execution of the product framework and how quickly a bug ought to be coordinated to the advancement group. Both scholar and business networks have made an expansive request to automate the bug seriousness expectation.

The information extracted from the bug reports plays a vital role in the development of the software, software evolution, and maintenance tasks. Bug reports are registered by the users in the response of their usage. Meanwhile, developers took the responsibility of extracting knowledge from these reports in order to evolve their systems effectively. There are several bug tracking systems (BTS) such as JIRA and Bugzilla, and they are considered as a bug report repository [3]. Generally, bug reports are often submitted by

the users through the bug tracking system. A bug report contains an appropriate circumstance under which an issue arises in an application and holds the information on error regeneration (shown in Figure 1). A general error report holds various aspects such as Bug ID, Status, Submission Date, Summary, Severity, Description, and Priority. However, among these attributes, the most critical attribute of the bug report is severity level of a bug, on the basis of which one decides how rapidly it should be resolved. Initially, the user initiates the process of assigning severity level to a bug, which may cause an incorrect assignment of severity due to less knowledge of the domain. Hence, the bug is reviewed by the software team representatives, and they decide to confirm or decline the severity level of a bug [4]. If a bug severity is confirmed by the software team, then its priority is defined and the bug is referred to a concerned individual to fix it. The severity evaluation of a bug is a manual process that requires a lot of effort and time. In order to carry out this task, a highly experienced and proficient person is required. Furthermore, there are a large number of bugs that are reported in software systems. The users notify a short statement about the severity level (e.g., critical, minor, major, blocker, and trivial). For example, an android project has opened 107,456 bugs from 2013 to 2015, whereas eclipse project had over 84,245 bug reports. Therefore, the manual evaluation of severity of bugs report is quite difficult and tedious task.

The impact of a bug differs from one client to another. Regularly, clients expect that their bug should be settled on need premise, no matter what its effect. Consequently, due to nonmindfulness, clients commit error while doling out the seriousness level to a bug [5]. The seriousness levels are fundamentally grouped into five general classifications, basic, minor, major, blocker, and unimportant, detailed as follows [2]:

(i) Unimportant: there are no time limitations for this level and it is an ostensible issue.

(ii) Minor: this level is minor and necessitates consideration yet with no particular time imperatives.

(iii) Major: this level of a bug needs consideration and should be settled soon. Demands for new highlights lie in this class.

(iv) Basic: this level of a bug is time-delicate and is problematic for the task yet without being troublesome to the essential usefulness of the venture.

(v) Blockers: this level of a bug is exceptionally time-touchy and is problematic for essential usefulness of the undertaking. It crashes the venture.

According to Umer et al. [6], emotions are involved in expressing the severity of bugs. Negative emotions of the end users elevate while reporting severe bugs as compared to nonseverity errors. For a severe bug, the terms or phrases used by most of the users are worst, pathetic, bad, and incorrect.

Kumari et al. [7] presented a technique to evaluate the severity level of bug reports. In this study, the authors used the data of irregularities and uncertainties. The Random Forest (RF), j48, K-Nearest Neighbor (KNN), Naïve Bayes (NB), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and MLR were used as a training classifier and ratify those using Mozilla, PITS, and Eclipse projects. The results indicated that their approach performed well as compared to the existing research.

There are several dimensions from where a developer can gain information to perform software evolution and maintenance activities such as regular inspection, reviews, and feedback from other developers and users. The knowledge discovered from the bug reports (submitted by users) is also considered as beneficial way to control the software evolution and assist the software construction.

Close to manual examination (by designers), computer programming research local area has utilized a few AI (ML) and Deep Learning (DL) techniques like CNN, K-NN, RF, RNN, NB, Naive Bayes Multinomial (NBM), SVM, and Decision Tree (DT) to robotize the bug report forecast at specific seriousness levels [8].

However, due to lack of capabilities of existing bug report severity predictors in terms of controlling the overfitting and simplicity in weight calculation (in the existence of data noise) and user's bias in terms of mentioning the wrong severity level (due to lack of enough domain knowledge) for bug report, there is a gap to introduce a new method to address these issues.

In order to control the overfitting and avoid the weight calculation complexity, we leverage the capabilities of GPT-2 while examining the user's bias; an emotion analysis is performed which includes an emotion score for each report during the bug report prediction process.

2. Related Work

Generally, bugs are classified as minor, major, trivial, and critical errors, where the trivial errors are not severe, while critical errors are the most severe for the users. Several ML and DL techniques are used to prearrange the severity of the bug reports.

Liu [9] predicted the severity of bug reports using data from Android test reports. A Knowledge Transfer Classification (KTC) method was presented based on machine learning and text mining. Liu's methodology utilized an Importance Degree Reduction (IDR) system dependent on unpleasant set to extricate trademark watchwords to get an increasingly exact reduction result.

Improving current feature selection technique through ranking-base strategy by Yang [10] used the severity bug reports from Eclipse and Mozilla. The results showed that the performance of the severity bug report was effectively improved; meanwhile, according to the results, the joint feature selection techniques performance was improved as compared to the single feature selection technique. Bug severity prediction in cross projects such as Wire Shark, Eclipse, Xamarine, and Mozilla was performed by Zhang [11]. Topic modeling was performed in cross projects using Latent Dirichlet Allocation (LDA). The proposed technique NBM was applied on cross projects and single projects, and the performance of cross projects was better than that of the single projects.

RETRACTED

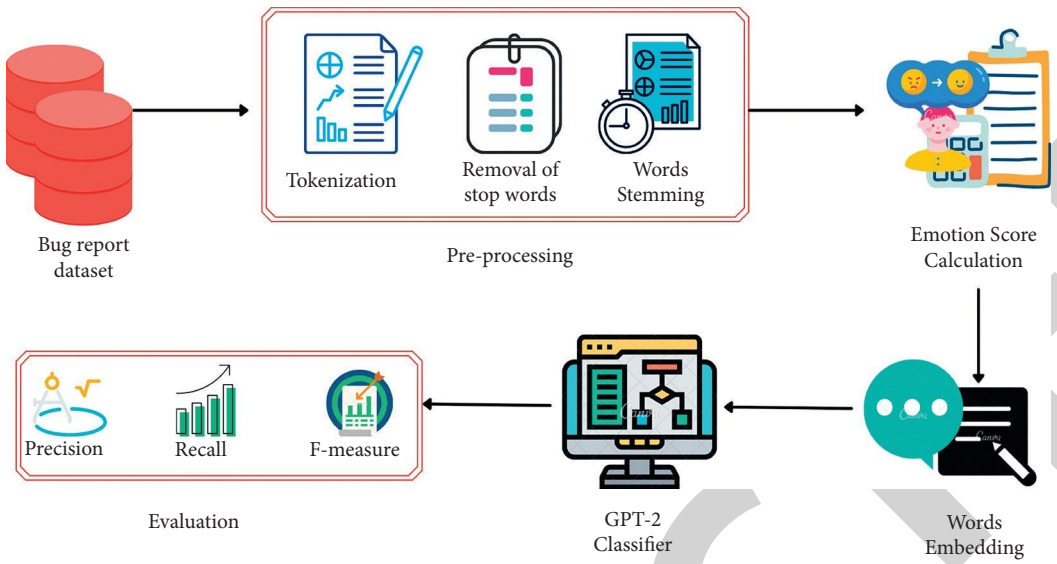


Figure 1: Proposed methodology.

Incremental learning approach was proposed by Zhang [12] for the bugs report recognition. A dynamic learning approach was established for tagging unlabelled bug reports; for instance, a sample augmentation approach was applied for the appropriate training data. Furthermore, various types of connectionism approach were used to detect bug reports. The complete set of experimentations on various bug report datasets validates that the generalisation capabilities of these models can be enhanced by this recommended incremental learning approach. In order to examine the effectiveness of different classification methods, Hamdy and El-Laithy [13] apply multiple ML classification approaches to find the severity of bug reports, and 29,204 bug reports were used for this prediction. Results indicated that their proposed model NBM outperformed the simple NB, SVM, and K-NN.

Similarly, Jindal [14] used oversampling approach "SMOTE" to equal the classes of severity and feature selection to select the informative features from the data and pass these features to K-KNN. That model was applied on two bug repositories (Mozilla and Eclipse). Results indicated that their model outperformed other models in predicting the minor severity. In another study, the bug reports dataset was extracted from various repositories such as PITS (PITS A to E) by Zhang [15]; these data repositories are used by the NASA engineers. They used statistical model Multinominal Multivariate Logistic Regression (MMLR), Decision Tree, and Multilayer Perceptron (MLP). The results indicated that Decision Tree outperformed the MMLR and MLP models. Modified REP and KNN techniques were proposed by Agrawal and Goyal [16] for the classification of the historical bug reports. They used the features of historical bug reports and applied these features on a new bug report. Their technique indicated better results.

Kannan [17] related Word2vec in helping designers to arrange bug seriousness level. This concentrate additionally investigates the impact of averaging procedures on classifier execution, particularly hyperparameter which further develops execution of classifiers on tuning suitably. The examination work additionally inspected the impact of window size and minimum count with three significance strategies for execution clearing of 6 classifiers which showed pertinence in bug seriousness. Notwithstanding, it negligibly affected bug examination and the utilization of averaging technique leaves trifling impact on execution.

Kumari and Singh [18] focused on further developing classifiers to foresee the bug report. They utilized Entropy and different methods, for example, NB and DL, to construct bug need classifiers. The review approved constructed classifiers utilizing 8 results of open office project. Entropy-based approach expanded and further developed precision up to various levels. The outcomes showed that utilization of Entropy-based approach gave more exact outcomes and really anticipated need of bug report. However, the review neglects it in other datasets and uses new methods other than entropy and, therefore, disregards information awkwardness.

Sabor et al. [19] presented a positioning-based future determination technique for execution enhancement of bug seriousness forecast as it saved investment. The exploration work tested the technique on web crawlers which showed that better execution and calculation gave better prediction of bug reports. In any case, the methodology utilized a misclassification work, which will in general lessen the exactness of forecast model and therefore, the ordinary articulation utilized in this review has missed some stack follows that also results decrease in precision. Moreover, for every product project there are a few sorts of bug report which the examination work neglects making sense of and gives just fundamental request of gathering calculation.

Ali et al. [20] carried out a parallel arrangement of programming occurrences. They involved dynamic learning and SVM to classify the product occurrences as disappointment or nondisappointment. The forecast model was gotten from the disappointment log documents from

RETRACTED

Obscuration. Their model accomplished a precision of 84%. Be that as it may, their review is restricted to paired grouping with restricted dataset.

2.1. Problem Identification and Solution. Analysts and experts have introduced different procedures to motorize the seriousness forecast of the bug reports. Generally, the conventional grouping approaches, for example, SVM, NB, and Decision Tree, are utilized to anticipate the seriousness level of a bug report. These customary methodologies limit the time utilization on assessment of bug seriousness reports and hoist the possibilities of precise order of bug seriousness. Be that as it may, because of absence of capacity to control the overfitting and the presence of information commotion during the weight computation, their exhibition turns into a challengeable errand and opens the entryway to improve the indicators either by presenting new strategies or by upgrading the current arrangements. Thus, we propose a philosophy to defeat these issues (information noising and overfitting) by utilizing GPT-2 classifier.

The fundamental commitment of this study is triple. Right off the bat, the creators address the issue of bug seriousness level forecast by controlling the overfitting, information clamor, and keeping away from weight estimation. Furthermore, a Deep Learning classifier with best hyperboundaries values is applied to accomplish most elevated exactness. Thirdly, the proposed classifier (GPT-2) gives the cutting-edge exactness.

3. Proposed Methodology

The proposed methodology is shown in Figure 1. The proposed methodology is comprised of 5 steps. Initially, the bug reports data of open-source projects is extracted from Mozilla and Eclipse history data. Afterwards the bug reports are preprocessed using natural language techniques. In the third step, an emotional score is calculated and allotted to each bug report. Afterwards, a vector (e.g., word embedding) is generated for each preprocessed bug report. Finally, the vector and emotional score are used to train a Deep Learning model in order to predict the severity level of the bug reports. The explanation of each step is given as follows.

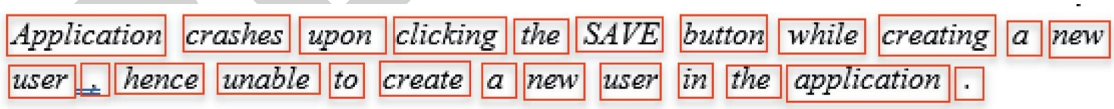
3.1. Data Extraction. The dataset was collected from the online forum of Bugzilla bug tracking system. We explored the bug database of Bugzilla to mine bug reports of Eclipse and Mozilla projects. The bug reports were extracted without any duplication. The extracted bug reports belong to seven open-source products, namely, CDT, Platform, JDT, Firefox, Bugzilla, Core, and Thunderbird. A total of 59616 bug reports were extracted, among which 16.77%, 8.39%, 16.77%, 16.77%, 7.76%, 16.77%, and 16.77% of bug reports belong to each product, respectively. The description of the dataset is given in Table 1.

3.2. Data Preprocessing. In data preprocessing, the data is encoded into a structured form. Data preprocessing is done in order to remove noise and garbage values from the dataset. It is basically cleaning of the data [21]. Hence, we perform several preprocessing steps to make the data useful for our proposed method. The set of preprocessing activities is shown in Figure 2 and the description is as follows.

3.2.1. Tokenization. Tokenization is a process to break the text down into smaller units known as tokens. The token can be a symbol, a word, a number, phrase, or other meaningful elements [22]. The list of these tokens is taken as an input for other preprocessing activities. In this activity, all the elements in the bug reports are broken down into tokens for further preprocessing.

For example, "*Application crashes upon clicking the SAVE button while creating a new the user, hence unable to create a new user in the application*."

The above statement extracted from the dataset is broken down as follows:



Here, all the elements of a sentence are highlighted as tokens. The sentence is broken down into 24 tokens. These tokens are now passed to the next phase for further processing.

3.2.2. Removal of Stop Words. Stop words are the words that are frequently used in the sentences and phrases (e.g., the, is, am, are, a, and an). Their removal is necessary as they create noise in the data and their existence can negatively affect the performance of the bug report severity level predictor [23]. As in the above mentioned example, some words such as "a," "the," "in," and "to" are stop words. These are meaningless and create data noise; hence, in this phase, these words are removed and passed for further processing.

3.2.3. Removal of Punctuation. This process is used to remove the punctuations (e.g., "$# %&'()∗+./: <=>?@[\]^_`{|}~) from the data. As the punctuations are meaningless elements in the bug reports, their removal is also necessary. Hence, in this activity, all the punctuations from the bug reports are removed, in order to minimize the noise and data sparsity from the data. As in the above example, the punctuations such as "." and "," are removed to reduce the data sparsity.

RETRACTED

Table 1: Description of the dataset.

Project	Description	Bugs reports	# of total bugs reports (%)
CDT	The C/C++ Development Toolkit (CDT) project provides Eclipse plug-ins. It is an open-source project.	5,031	16.77
Platform	The Eclipse platform is developed for building integrated development environments (IDEs) and arbitrary tool.	2,517	8.39
JDT	The JDT project offers the capabilities of full-feature plug-ins of Java IDE.	5,031	16.77
Firefox	Mozilla foundation developed a free open-source web browser, Firefox. It is officially available for MS Windows, Linux, and macOS.	5,031	16.77
Bugzilla	Bugzilla is an issue/bug tracking system to allow an individual or a bunch of experts to keep records of bugs with their product.	2,328	7.76
Core	Core basically is the shared mechanisms used by Mozilla software, containing handling of web content (HTML, Layout, CSS, etc.)	5,031	16.77
Thunderbird	Mozilla foundation developed a free open-source news client, chat client, RSS, and cross-platform e-mail client, named Thunderbird.	5,031	16.77

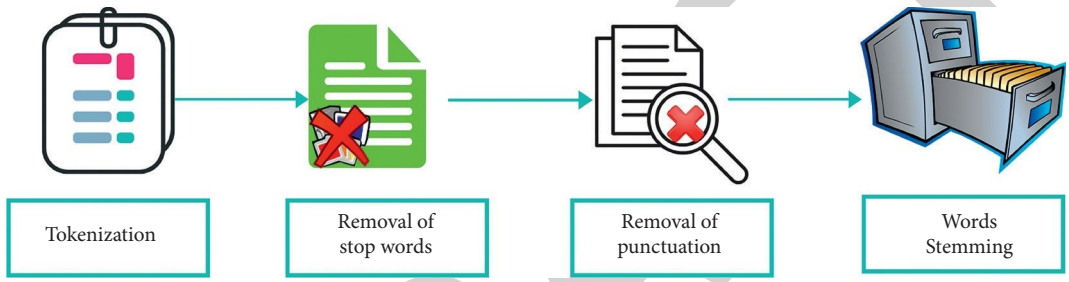


Figure 2: Data preprocessing.

3.2.4. Word Stemming. Word stemming is the fourth activity performed for the preprocessing of bug reports. In this process, each word is compressed into its base or a common root [24]. It is basically cutting off the suffixes or prefixes of the word and changing it into an inflected word [25]. Hence, all the words in the bug reports are compressed to their common roots, respectively. For example, in the above example, the word "creating" is compressed to its root word "create"; similarly, the word "clicking" is compressed to "click."

After completing the preprocessing steps, the data is to be converted into vectors.

3.3. Emotion Score Calculation. Nowadays, researchers are trying to work on making the machines emotionally intelligent. The text data (in our case it contains bug reports) in software engineering may consist of emotional phrases or words regarding the software system. For example, the use of positive words (e.g., well, better, and fine) in a bug report may lead to a positive emotion of the user. Similarly, the use of negative words (e.g., wrong, bad, and suffer) in a bug report may indicate negative emotion of user. In our case, the use of words (i.e., "wrong," "break," and "incorrect") indicates the highest severity level of bug reports. Subsequently, the use of words (i.e., "minor" and "little") indicates the lowest severity level of bug reports. For the emotional score calculation, we use Senti4SD database [26], as it is a widely used database for the emotion analysis in organizing unstructured data in several domains of software engineering.

3.4. Words Embedding. Deep Learning-based classifiers need to transform textual data of bug reports into fixed-length numerical vectors using Word2vec technique [27]. Word2vec is an effective technique for learning great feature distributed vector representation. In this stage, each preprocessed bug report is passed to Word2vec, which transforms each preprocessed bug report into a fixed-length numerical vector. This vector model is then passed to the classifier for the prediction of bug severity level.

3.5. GPT-2. GPT-2 (Generative Pretrained Transformer-2) is an open-source transformer model that was proposed in 2019 by Open AI. It is a scale-up model of GPT [28]. It implements a deep neural net using attention mechanism, which focuses on most relevant segments of the input text. It uses transformer decoder blocks [29]. For the classification of bug report severity level, we employed a Deep Learning-based model named GPT-2. We leverage the capabilities of GPT-2 owing to its important features: (1) It leads to the regularization by controlling the overfitting issue. (2) It starts from a weak model and outcomes a strong model. (3) Its execution speed and model performance have been empirically investigated. (4) It works as a series of predictors for removing errors in each predictor and gives a new predictor with the aim of minimum loss as compared to the previous predictor. Although GPT-2 is available as a pre-trained model, in this study, we have trained the GPT-2 independently. GPT-2 showed a remarkable performance in several domains such as protein and genes prediction [30],

RETRACTED

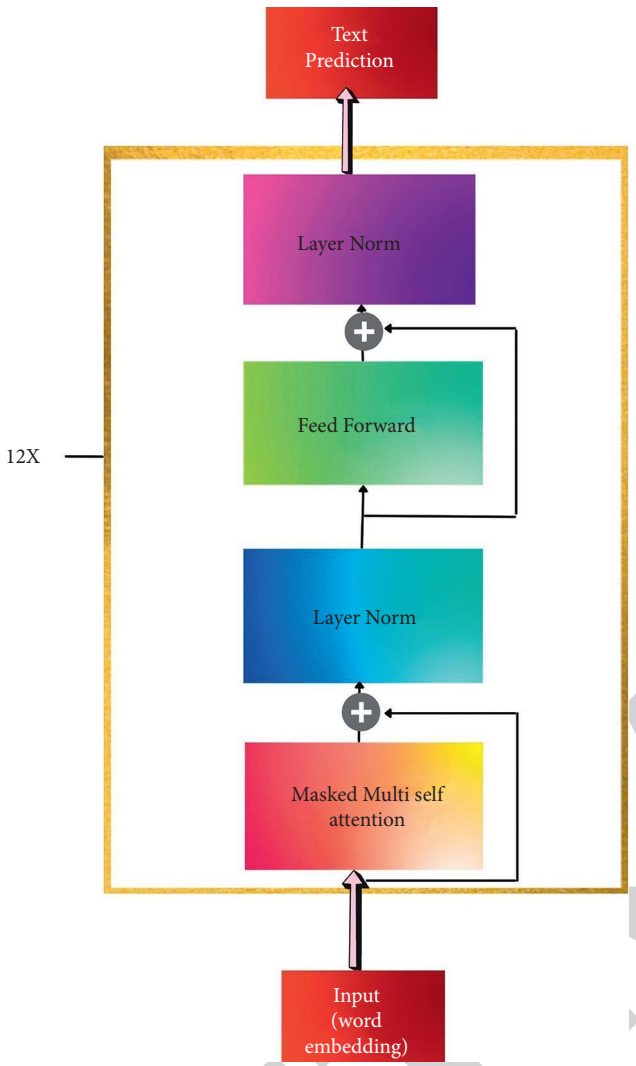


Figure 3: GPT-2 architecture.

nontyphoidal *Salmonella* genomes [31], classifying spam reviews [32], and personalized recipe generation [33].

The GPT configuration uses consideration rather than earlier repeat and convolution-based structures to make a profound brain organization, explicitly called a transformer model. The model's attention processes permit it to zero specifically in on segments of information text that i.e. the input it accepts are the most significant. This model considers substantially more parallelization and beats prior CNN/RNN/LSTM-based model benchmarks. Figure 3 presents the engineering of GPT-2.

There are 12 layers of transformer. Each layer of transformer is comprised of 12 free consideration instruments. These consideration systems are known as "heads." Thus, there are $12 \times 12 = 144$ consideration designs. Every one of the consideration designs gains significant component from the installing and utilizes the learned examples to foresee the class. There are 7 classes: blocker, basic, major, minor, improvement, typical, and trifling. Henceforth, the consideration component predicts classes for each info inserting (Algorithm 1).

4. Evaluation Model

To evaluate the efficacy and performance of the proposed method, we employed widely known evaluation metrics, such as accuracy, recall, precision, and *F*1-score [34].

The accuracy describes how close the measured value is to the known value [35]. The formula to calculate accuracy is

$$\textbf{Accuracy} = \frac{(\text{TP} + \text{TN})}{(\text{TP} + \text{FP} + \text{FN} + \text{FN})}. \quad (1)$$

In the above formula, TP is True Positive, TN is True Negative, FP is False Positive, and FN is False Negative.

Another type of statistical measure used is recall which describes the sensitivity of the ML model [36]. It is defined as the ratio of accurately predicted positive samples to the total number of the observations. The formula to calculate recall is

$$\textbf{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \quad (2)$$

In the above formula, TP is True Positive and FN is False Negative.

Precision is the ratio of accurately predicted positive samples to the total number of positive predictive samples [35]. It is calculated as

$$\textbf{Precision} = \frac{\text{TP}}{(\text{TP} + \text{FP})}. \quad (3)$$

In the above formula, TP is True Positive and FP is False Positive.

Another standard statistical measure for the ML classifiers is *F*1-score. It is defined as the average of precision and recall. It is the harmonic mean of recall and precision. The formula to calculate *F*1-score is

$$\textbf{F1 Score} = 2 * \frac{(R * P)}{(R + P)}. \quad (4)$$

In the above formula, R is recall and P is precision.

In this study, we compared the bug severity predictors on the basis of *F*1-score, as it is the mean of both recall and precision. So, the value of *F*1-score plays a vital role in analyzing the effectiveness of our proposed model and comparing the performances of the classifiers.

5. Result and Discussion

For the investigations, we utilized 10-crease cross-approval method; for example, in the principal stage, we use 90% of the dataset for preparing the model and 10% for testing the model. In the second stage, 80% of the dataset was utilized for preparing the model and 20% for testing the model. Also, in the third stage, 70% of the information was utilized for preparing the model and 30% for testing. In the following emphasis, 60% of the information was utilized for the preparation of the model, while 40% was utilized for testing the model. Furthermore, 10 cross fold validation was performed to validate the model. We have utilized Google Colab for the execution of the model. Gadget = torch.device(); work was utilized to search for GPU to utilize. Name and

REJECTED

Begin:
Preprocessing of extracted data
Input: Statements S extracted from N Bug reports.
(a) Removal of A = stop words from Statements S
(b) Removal of P = punctuation from Statements S
(c) Perform word stemming ($w \longrightarrow$ root word) in the statements S
Calculate emotion score E using senti4SD
Calculate word embedding for each vector V_{o_i} using the equation:
$P(w_c|W_o) = (\exp(U_c{}^T \underline{v}_o)/\sum^{i\epsilon v} \exp(U_i{}^T \underline{v}_o))$
Where, T is length of text sequence, $W_o = w_{o1}, w_{o2}, \ldots . w_{2m}$ and $(\underline{v}_o = \underline{v}_{o1} + \underline{v}_{o2} + \underline{v}_{o3}, \ldots . \underline{v}_{o2m}/2m)$.
Pass the embedded vectors to the classifier GPT-2 such that:
T = training data
X = pool of examples by choosing randomly
For $i = 0$ to K
Use T for training the classifier, 80%
Use remaining 20% of T to test the classifier
End for end

Algorithm 1: Pseudocode of the proposed methodology.

way of transformer model were given utilizing capacity (model_name_or_path = "gpt2"). The tuning of the boundaries of the model was done to accomplish the cutting-edge precision, for example,

Set_seed (123); //set seed for reproducibility

Epochs = 16; //Number of training epochs

Batch_size = 32; //Number of batches

Max_length = 60; //truncate text sequence

Classes = 7; //Number of classes

Label_ids = {'blocker': 0, 'Critical': 1, 'Major': 2, 'Minor': 3, 'Enhancement':4, 'Normal': 5, 'Trivial': 6}; //Dictionary of labels with ids, String labels with ids.

Top_K; //This boundary controls the variety. In the event that its worth is set to 1, it implies just 1 word is considered at each progression, so to keep away from the time intricacy we have utilized $K = 40$, 40 tokens at each progression, by and large it is a decent worth.

We have presented our preliminary results in two aspects. Firstly, we analyzed the effectiveness of the proposed method in terms of prediction of appropriate severity level of bug report. The results are shown in Table 1. Secondly, in order to assess the effectiveness of proposed method from the state-of-the-art bug report predictor, we perform several experiments. The results are presented in Table 2 and Figure 4.

There were 7 classes of severity level: blocker, enhancement, critical, major, trivial, minor, and normal). The recall, $F1$-score, and precision of each class are given in Table 2. For the bug severity predictor, we are more concerned about the value of $F1$-score as it the harmonic ratio of both recall and precision. The consequences of the results shown in Algorithm 1 are as follows:

(i) In terms of $F1$-score, the proposed method outperforms the others in predicting bug report, which is categorized as "minor" severity level (100%).

(ii) In terms of $F1$-score, the performance of the proposed method in predicting bug report associated with "normal" severity level remained lowest (82%) as compared to the rest of the severity levels.

(iii) The overall performance of the proposed method at all severity levels remained better, which indicates its effectiveness in terms of predicting the severity levels of the bug reports.

For the second aspect, the proposed method was compared with state-of-the-art techniques, and the results are shown in Table 3, where, in terms of $F1$-score, we can observe the better performance of the proposed method as compared to the state-of-the-art techniques. The proposed method with $F1$-score value of 91% indicates that it outperforms CNN ($F1$-score of 54.91%), LSTM ($F1$-score of 75.88%), NB ($F1$-score of 47.31%), RF ($F1$-score of 89.55%), and SVM ($F1$-score of 48.45%). This can also be observed in Figure 4. Besides, as we carried out the near investigation of GPT-2 with different classifiers, the outcomes are displayed in Tables 4–9.

Table 4 presents the assessment of CNN as regards precision, recall, and $F1$-score for each of the 7 classes of the bug, where, for blocker, precision, recall, and $F1$-score are 0.65, 0.81, and 0.72, for critical, they are 0.40, 0.51, and 0.45, for enhancement, they are 0.91, 0.96, and 0.94, for major, they are 0.49, 0.04, and 0.08, for minor, they are 0.49, 0.29, and 0.36, for normal, they are 0.53, 0.77, and 0.62, and for trivial, they are 0.54, 0.66, and 0.59.

Table 5 presents the exhibition assessment of LSTM as regards precision, recall, and $F1$-score for each of the 7 classes of the bug, where, for blocker, precision, recall, and $F1$-score are 0.84, 0.95, and 0.89, for critical, they are 0.55, 0.63, and 0.58, for enhancement, they are 0.94, 0.99, and 0.97, for major, they are 0.51, 0.30, and 0.38, for minor, they are 0.68, 0.56, and 0.62, for normal, they are 0.64, 0.77, and 0.70, and for trivial, they are 0.79, 0.83, and 0.81.

RETRACTED

Table 2: Performance evaluation of GPT-2.

Class	Precision (%)	Recall (%)	*F*1-score (%)
Blocker	98	100	99
Critical	99	91	95
Enhancement	100	95	97
Major	99	95	97
Minor	99	100	100
Normal	99	70	82
Trivial	82	100	90

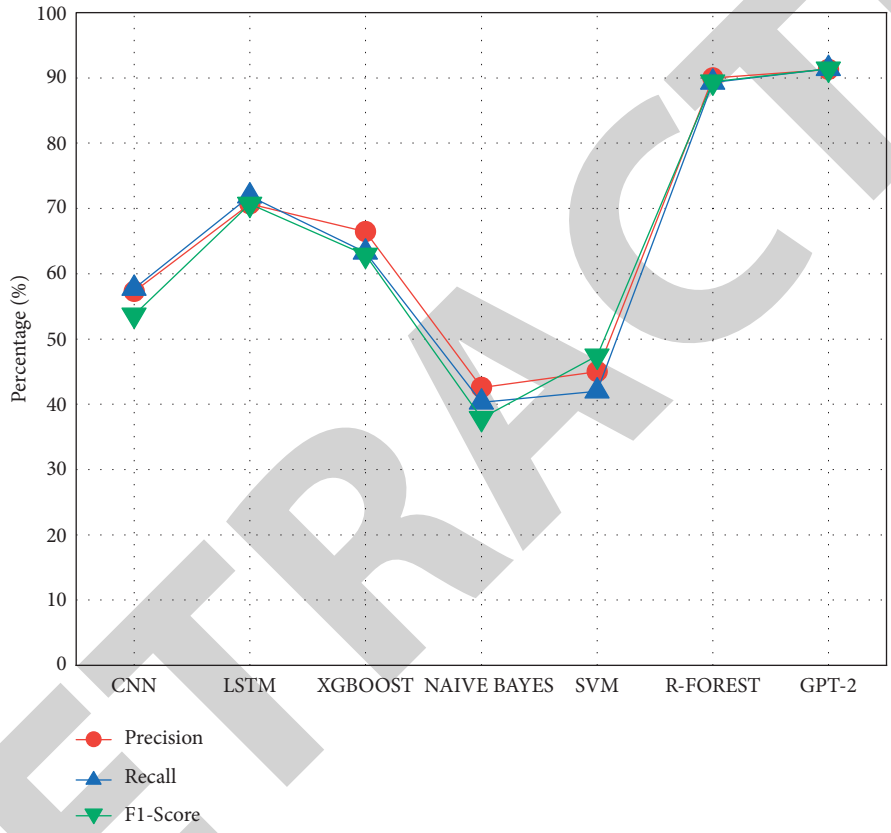


Figure 4: Comparative analysis of proposed technique with state-of-the-art techniques.

Table 3: Comparative analysis of the proposed method with CNN, LSTM, NB, RF, and SVM.

Classifier	Accuracy (%)	Precision (%)	Recall (%)	*F*1-score (%)
Proposed method (based on GPT-2)	**91.41**	**93.00**	**91.00**	**91.00**
CNN	75.74	58.93	57.32	54.91
LSTM	81.18	70.15	71.48	75.88
NB	50.73	45.54	42.74	47.31
RF	90.71	89.22	88.11	89.55
SVM	52.89	47.34	45.55	48.45

Table 6 presents the exhibition assessment of NB as regards precision, recall, and *F*1-score for each of the 7 classes of the bug, where, for blocker, precision, recall, and *F*1-score are 0.57, 0.23, and 0.33, for critical, they are 0.36, 0.55, and 0.44, for enhancement, they are 0.69, 0.67, and 0.68, for major, they are 0.27, 0.05, and 0.09, for minor, they are 0.28, 0.25, and 0.27, for normal, they are 0.40, 0.84, and 0.54, and for trivial, they are 0.41, 0.23, and 0.30.

Table 7 presents the assessment of RF as regards precision, recall, and *F*1-score for each of the 7 classes of the bug, where, for blocker, precision, recall, and *F*1-score are 0.95, 0.94, and 0.95, for critical, they are 0.71, 0.86, and 0.78,

RETRACTED

Table 4: Performance evaluation of CNN.

Class	CNN		
	Precision	Recall	*F*1-score
Blocker	0.65	0.81	0.72
Critical	0.40	0.51	0.45
Enhancement	0.91	0.96	0.94
Major	0.49	0.04	0.08
Minor	0.49	0.29	0.36
Normal	0.53	0.77	0.62
Trivial	0.54	0.66	0.59

Table 5: Performance evaluation of LSTM.

Class	LSTM		
	Precision	Recall	*F*1-score
Blocker	0.84	0.95	0.89
Critical	0.55	0.63	0.58
Enhancement	0.94	0.99	0.97
Major	0.51	0.30	0.38
Minor	0.68	0.56	0.62
Normal	0.64	0.77	0.70
Trivial	0.79	0.83	0.81

for enhancement, they are 0.98, 0.99, and 0.99, for major, they are 0.89, 0.77, and 0.82, for minor, they are 0.90, 0.87, and 0.88, for normal, they are 0.93, 0.89, and 0.91, and for trivial, they are 0.94, 0.93, and 0.93.

Table 8 presents the exhibition assessment of SVM as regards precision, recall, and *F*1-score for each of the 7 classes of the bug, where, for blocker, precision, recall, and *F*1-score are 0.52, 0.34, and 0.41, for critical, they are 0.33, 0.67, and 0.44, for enhancement, they are 0.74, 0.87, and 0.80, for major, they are 0.27, 0.06, and 0.10, for minor, they are 0.32, 0.23, and 0.26, for normal, they are 0.50, 0.74, and 0.50, and for trivial, they are 0.47, 0.23, and 0.31.

Table 9 presents the exhibition assessment of XGBoost as regards precision, recall, and *F*1-score for each of the 7 classes of the bug, where, for blocker, precision, recall, and *F*1-score are 0.85, 0.74, and 0.79, for critical, they are 0.43, 0.69, and 0.53, for enhancement, they are 0.93, 0.97, and 0.95, for major, they are 0.57, 0.23, and 0.33, for minor, they are 0.56, 0.45, and 0.50, for normal, they are 0.57, 0.79, and 0.66, and for trivial, they are 0.74, 0.56, and 0.64.

In the figure 4 above the performance percentage in terms of precision, f-measure and recall for each of the technique are given. Where, our model "GPT-2" outperformed the rest of the models. The CNN model had following values in terms of F-measure, for each of the severity level class (Blocker (0.72), Critical (0.45), major (0.94), minor (0.08), normal (0.36), trivial (0.62), and enhancement (0.59). For LSTM, the values for each of the severity level class are (Blocker (0.89), Critical (0.58), major (0.97), minor (0.38), normal (0.62), trivial (0.70), and enhancement (0.81). For XGBoost, the values for each of the severity level class are (Blocker (0.79), Critical (0.53), major (0.95), minor (0.33), normal (0.50), trivial (0.66), and enhancement (0.64). For Naïve bayes, the values for each of the severity level class are (Blocker (0.33), Critical (0.44), major (0.68), minor (0.09), normal (0.27), trivial (0.54), and enhancement (0.30). For SVM, the values for each of the severity level class are (Blocker (0.41), Critical (0.44), major (0.80), minor (0.10), normal (0.26), trivial (0.50), and enhancement (0.31). For Random Forest, the values for each of the severity level class are (Blocker (0.95), Critical (0.78), major (0.99), minor (0.82), normal (0.88), trivial (0.91), and enhancement (0.93). Similarly, for the proposed model "GPT-2", the values for each of the severity level class are (Blocker (0.99), Critical (0.95), major (0.97), minor (0.97), normal (100), trivial (0.82), and enhancement (0.90).

Table 6: Performance evaluation of NB.

Class	NB		
	Precision	Recall	*F*1-score
Blocker	0.57	0.23	0.33
Critical	0.36	0.55	0.44
Enhancement	0.69	0.67	0.68
Major	0.27	0.05	0.09
Minor	0.28	0.25	0.27
Normal	0.40	0.84	0.54
Trivial	0.41	0.23	0.30

Table 7: Performance evaluation of RF.

Class	Random Forest		
	Precision	Recall	*F*1-score
Blocker	0.95	0.94	0.95
Critical	0.71	0.86	0.78
Enhancement	0.98	0.99	0.99
Major	0.89	0.77	0.82
Minor	0.90	0.87	0.88
Normal	0.93	0.89	0.91
Trivial	0.94	0.93	0.93

Table 8: Performance evaluation of SVM.

Class	SVM		
	Precision	Recall	*F*1-score
Blocker	0.52	0.34	0.41
Critical	0.33	0.67	0.44
Enhancement	0.74	0.87	0.80
Major	0.27	0.06	0.10
Minor	0.32	0.23	0.26
Normal	0.50	0.74	0.50
Trivial	0.47	0.23	0.31

Table 9: Performance evaluation of XGBoost.

Class	XGBoost		
	Precision	Recall	*F*1-score
Blocker	0.85	0.74	0.79
Critical	0.43	0.69	0.53
Enhancement	0.93	0.97	0.95
Major	0.57	0.23	0.33
Minor	0.56	0.45	0.50
Normal	0.57	0.79	0.66
Trivial	0.74	0.56	0.64

RETRACTED

6. Conclusion

Bug severity level is one of the major parameters of bug reports to analyze the quality of software development. There are several processes to classify the bugs on the basis of their severity level. But, due to time consumption, inaccuracy, overfitting, and the existence of data noise, this task is still challenging. Hence, we propose a new methodology to classify the bugs on the basis of their severity level. The proposed method is functional in four phases and the bug report dataset is extracted from a repository (Bugzilla). Firstly, we apply the text preprocessing of bug reports using natural language processing (NLP) techniques. Secondly, we perform an emotion analysis on each bug report and compute an emotion score. Thirdly, we generate a feature vector for each report. Finally, we use the emotion score and vectors of all reports as an input of GPT-2. Afterwards, the performance of the proposed method is evaluated using the standard evaluation metrics; furthermore the employed classifier GPT-2 is compared with the state-of-the-art methods such as CNN, LSTM, NBM, and RF. The promising results (91.4% accuracy) of the GPT-2 classifier indicated that it is the most suitable machine learning classifier for classifying the severity level of bugs.

For the future work, we can include more classifiers and datasets for the empirical investigation of predicting bug severity level.

Data Availability

The data used to support the findings of this study are collected from online sources and will be available from the corresponding authors upon request.

Conflicts of Interest

The authors declare no potential conflicts of interest.

Authors' Contributions

The first author contributed to conceptualization, methodology, data curation, visualization, investigation, and writing, and the rest of the authors contributed to supervision, reviewing, and revision.

References

[1] E. Giger, P. Martin, and H. Gall, "Predicting the fix time of bugs," in *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, May 2010.

[2] L. A. F. Gomes, R. D. S. Torres, and M. L. Côrtes, "Bug report severity level prediction in open source software: a survey and research opportunities," *Information and Software Technology*, vol. 115, pp. 58–78, 2019.

[3] Z. Li, P. Liang, D. Li, R. Mo, and B. Li, "Is bug severity in line with bug fixing change complexity?" *International Journal of Software Engineering and Knowledge Engineering*, vol. 30, no. 11n12, pp. 1779–1800, 2020.

[4] K. K. Chaturvedi and V. B. Singh, "An empirical comparison of machine learning techniques in predicting the bug severity of open and closed source projects," *International Journal of Open Source Software & Processes*, vol. 4, no. 2, pp. 32–59, 2012.

[5] A. Baarah, "Machine learning approaches for predicting the severity level of software bug reports in closed source projects," *International Journal of Advanced Computer Science and Applications*, vol. 10, Article ID 14569, 2019.

[6] Q. Umer, H. Liu, and Y. Sultan, "Emotion based automated priority prediction for bug reports," *IEEE Access*, vol. 6, pp. 35743–35752, 2018.

[7] M. Kumari, M. Sharma, and V. B. Singh, "Severity assessment of a reported bug by considering its uncertainty and irregular state," *International Journal of Open Source Software & Processes*, vol. 9, no. 4, pp. 20–46, 2018.

[8] S. Guo, R. Chen, and H. Li, "Using knowledge transfer and rough set to predict the severity of android test reports via text mining," *Symmetry*, vol. 9, no. 8, p. 161, 2017.

[9] W. Liu, "Predicting the severity of bug reports based on feature selection," *International Journal of Software Engineering and Knowledge Engineering*, vol. 28, pp. 537–558, 2018.

[10] G. Yang, "Applying topic modeling and similarity for predicting bug severity in cross projects," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 13, pp. 1583–1598, 2019.

[11] T.-L. Zhang, "An uncertainty based incremental learning for identifying the severity of bug report," *International Journal of Machine Learning and Cybernetics*, vol. 11, pp. 123–136, 2020.

[12] T. Zhang, G. Yang, B. Lee, and A. T. S. Chan, "Predicting severity of bug report by mining bug repository with concept profile," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pp. 1553–1558, New York, NY, USA, April 2015.

[13] A. Hamdy and A. El-Laithy, "Smote and feature selection for more effective bug severity prediction," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, pp. 897–919, 2019.

[14] R. Jindal, R. Malhotra, and A. Jain, "Prediction of defect severity by mining software project reports," *International Journal of System Assurance Engineering and Management*, vol. 8, pp. 334–351, 2017.

[15] W. Zhang, S. Wang, and Q. Wang, "KSAP: an approach to bug report assignment using KNN search and heterogeneous proximity," *Information and Software Technology*, vol. 70, pp. 68–84, 2016.

[16] R. Agrawal and R. Goyal, "Developing bug severity prediction models using word2vec," *International Journal of Cognitive Computing in Engineering*, vol. 2, pp. 104–115, 2021.

[17] S. Kannan, "Preprocessing techniques for text mining," *International Journal of Computer Science & Communication Networks*, vol. 5, pp. 7–16, 2014.

[18] M. Kumari and V. B. Singh, "An improved classifier based on entropy and deep learning for bug priority prediction," in *Proceedings of the International Conference on Intelligent Systems Design and Applications*, Springer, Vellore, India, 2018.

[19] K. K. Sabor, M. Hamdaqa, and L. AbdelwahabHamou, "Automatic prediction of the severity of bugs using stack traces and categorical features," *Information and Software Technology*, vol. 123, Article ID 106205, 2020.

[20] S. Ali, M. Adeel, SumairaJohar, M. Zeeshan, S. Baseer, and A. Irshad, "Classification and Prediction of Software Incidents Using Machine Learning Techniques," *Security and Communication Networks*, vol. 2021, Article ID 9609823, 16 pages, 2021.

RETRACTED

[21] N. Novielli, F. Calefato, and F. Lanubile, "Love, joy, anger, sadness, fear, and surprise: Se needs special kinds of ai: a case study on text mining and se," *IEEE Software*, vol. 37, pp. 86–91, 2020.

[22] Y. Goldberg and O. Levy, "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method," (2014), https://arxiv.org/abs/1402.3722.

[23] P. Budzianowski and I. Vulić, "Hello, it's GPT-2--how can I help you? towards the use of pretrained language models for task-oriented dialogue systems," 2019, https://arxiv.org/abs/1907.05774.

[24] K. Ethayarajh, "How contextual are contextualized word representations? comparing the geometry of BERT, ELMo, and GPT-2 embeddings," 2019, https://arxiv.org/abs/1909.00512.

[25] V. Kieuvongngam, B. Tan, and YimingNiu, "Automatic text summarization of covid-19 medical research articles using bert and gpt-2," 2020, https://arxiv.org/abs/2006.01997.

[26] H. Zhou, "Improving neural protein-protein interaction extraction with knowledge selection," *Computational Biology and Chemistry*, vol. 83, Article ID 107146, 2019.

[27] S. A. Alvarez, "An exact analytical relation among recall, precision, and classification accuracy in information retrieval," Technical Report BCCS-02-01, pp. 1–22, Boston College, Boston, USA, 2002.

[28] Webster, J. Jonathan, and K. Chunyu, "Tokenization as the initial phase in NLP," in *Proceedings of the 14th International Conference on Computational Linguistics*, Samos, Greece, 1992.

[29] S. Hussain, A. Nasir, K. Aslam, S. Tariq, and M. F. Ullah, "Predicting mental illness using social media posts and comments," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 12, 2020.

[30] J. A. Bullinaria and J. P. Levy, "Extracting semantic representations from word co-occurrence statistics: stop-lists, stemming, and SVD," *Behavior Research Methods*, vol. 44, pp. 890–907, 2012.

[31] J. Singh and V. Gupta, "A systematic review of text stemming techniques," *Artificial Intelligence Review*, vol. 48, pp. 157–217, 2017.

[32] A. A. Irissappane, "Leveraging GPT-2 for classifying spam reviews with limited labeled data via adversarial training," 2020, https://arxiv.org/pdf/2012.13400.pdf.

[33] S. Ali, S. Baseer, I. A. Abbasi, B. Alouffi, W. Alosaimi, and J. Huang, "Analyzing the interactions among factors affecting cloud adoption for software testing: a two-stage ISM-ANN approach," *Soft Computing*, vol. 26, no. 4, 2022.

[34] M. Hossin, M. Sulaiman, and R. Wirza, "Improving accuracy metric with precision and recall metrics for optimizing stochastic classifier," in *Proceedings of the 3rd International Conference on Computing and Informatics (ICOCI 2011)*, Bandung, Indonesia, 2011.

[35] N. Tatbul, "Precision and recall for time series," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[36] M. Junker, R. Hoch, and A. Dengel, "On the evaluation of document analysis components by recall, precision, and accuracy," in *Proceedings of the Fifth International Conference on Document Analysis and Recognition ICDAR'99 (Cat. No. PR00318)*, IEEE, Bangalore, India, September 1999.

RETRACTED