The Journal of Systems and Software 208 (2024) 111898

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

# BERT based severity prediction of bug reports for the maintenance of mobile applications[☆]

Asif Ali[a], Yuanqing Xia[a,*], Qasim Umer[b,c], Mohamed Osman[a]

[a] School of Automation, Beijing Institute of Technology, Beijing 100081, China
[b] Department of Computer Science, Hanyang University, Seoul, 04763, Republic of Korea
[c] Department of Computer Sciences, COMSATS University Islamabad, Vehari 61000, Pakistan

ARTICLE INFO

*Keywords:*
Classification
BERT
Deep learning
User feedback
Mobile app reviews
Reliability

ABSTRACT

Mobile application maintenance is crucial to ensuring the accurate operation and continuous improvement of mobile applications (mobile apps). To effectively address issues and enhance the user experience, developers utilize issue-tracking systems that gather bug reports to refine mobile apps. Users can submit bugs through these systems, allowing them to determine the severity of each reported issue. The severity level plays a pivotal role in prioritizing bug resolution, enabling developers to address critical bugs promptly. Nonetheless, manually assessing the severity of each issue can be laborious and prone to errors. To overcome this challenge, this paper presents Bidirectional Encoder Representations from Transformers (BERT) based severity prediction of bug reports (called BERT-SBR) that leverages a deep neural network for automatic bug severity classification for mobile app maintenance. We collect the publicly available mobile apps bug reports dataset from the Hugging Face. BERT-SBR first computes the sentiment of reporters of bug reports and preprocesses them by leveraging BertTokenizer input formatting techniques. Next, it passes the formatted text and computed sentiment of each bug report to generate word embeddings. Then, it introduces a fine-tuned BERT classifier for bug report severity prediction. After that, it passes the generated word embeddings to the fine-tuned BERT classifier for training and testing. Finally, the proposed classifier's performance is evaluated. The BERT-SBR assessment results confirm that the fine-tuned BERT classifies bug reports significantly more effectively than other deep learning classifiers. On average, BERT-SBR achieves a remarkable improvement of 40.43%, 67.78%, 40.71%, and 58.14% in the accuracy, precision, recall, and f-measure. This indicates its superiority in accurately predicting the severity of bug reports for mobile application maintenance.

## 1. Introduction

Mobile application maintenance plays a crucial role in the overall mobile application development process. One of the key aspects of maintenance is effectively resolving bugs encountered by users while using released mobile applications (mobile apps) (Moran, 2015). Collecting and managing bug reports is vital for developers aiming to enhance mobile apps. Bug tracking systems, i.e., JIRA (Atlassian, 2018), are commonly utilized by developers to streamline bug reporting and bug triaging processes (Xia et al., 2013).

Users typically submit bug reports through issue tracking systems, providing detailed information about mobile app bugs and the circumstances leading to their occurrence. A typical bug report includes various attributes, i.e., bug ID, submission date, severity, summary, status, priority, and description. The severity attribute of a bug report is particularly significant, as it determines the urgency and priority of bug resolution. In systems like Mozilla (2018), severity can range from "trivial and "minor" to "normal", "major", "critical", and "blocker". Users manually assign severity during the reporting process, which can be a tedious task requiring domain knowledge. Incorrect severity assessment can occur due to user inexperience.

Researchers have proposed different approaches (Ramay et al., 2019; Sharma et al., 2014; Chaturvedi and Singh, 2012; Lamkanfi et al., 2011; Menzies and Marcus, 2008; Lamkanfi et al., 2010; Tian et al., 2012) to automate severity prediction in bug reports. Many of these approaches rely on traditional machine/deep learning classification algorithms. However, these approaches still need significant improvement, and none consider implicitly trained word embeddings. Moreover, reporters' sentiment has a significant influence on severity prediction (Ramay et al., 2019; Umer et al., 2018). Umer et al. (2018) reported that severe bugs tend to have a higher count of negative

☆ Editor: Aldeida Aleti.
* Corresponding author.
*E-mail addresses:* 3820192003@bit.edu.cn (A. Ali), xia_yuanqing@bit.edu.cn (Y. Xia), qasimumer@hanyang.ac.kr, qasimumer@cuivehari.edu.pk (Q. Umer).

https://doi.org/10.1016/j.jss.2023.111898
Received 28 August 2023; Received in revised form 22 October 2023; Accepted 3 November 2023
Available online 7 November 2023

sentiment expressed by reporters than non-severe bugs. Reporters often express their views while writing bug reports. For instance, the usage of words/phrases like *pathetic interface*, *dark*, and *problematic* by reporters may indicate the urgency of a bug report. Incorporating such sentiments could aid in the severity prediction of bug reports for mobile apps.

To address such challenges, this paper presents Bidirectional Encoder Representations from Transformers (BERT) based severity prediction of bug reports (called BERT-SBR). BERT-SBR leverages a deep neural network to predict the severity of bug reports for mobile app maintenance. The BERT-SBR begins by computing bug reports' sentiment and preprocessing them through BertTokenizer input formatting techniques. It then generates word embeddings for each preprocessed bug report. A fine-tuned BERT classifier is introduced to predict bug severity. The generated word embeddings are passed to the fine-tuned BERT classifier for training and testing. Finally, the proposed classifier's performance is evaluated. Evaluation results of BERT-SBR demonstrate that the fine-tuned BERT classifier significantly outperforms other deep learning classifiers in classifying bug reports.

The contributions of this study can be highlighted as :

- We present an innovative approach utilizing BERT for predicting bug report severity. As far as our knowledge extends, this is the initial implementation of BERT for predicting the severity of bug reports for mobile app maintenance.
- BERT-SBR achieves a remarkable improvement of 40.43% in accuracy, 67.78% in precision, 40.71% in recall, and 58.14% in f-measure. These results indicate the superiority of BERT-SBR in accurately predicting the severity of bug reports for mobile app maintenance.

The remaining sections of this study are organized as follows: Section 2 explores related work in the field. Section 3 provides a detailed explanation of BERT-SBR. Section 4 describes the evaluation process and presents the results. Section 5 discusses potential threats to the study. Finally, Section 6 concludes the paper and outlines potential avenues for future research.

## 2. Literature review

Bug reports are commonly classified based on severity levels, ranging from critical to trivial. Critical bugs are the most severe, while trivial bugs only cause minor inconvenience to users. To determine the priority of bug reports, researchers have proposed various machine/deep learning methods.

### *2.1. Machine learning based severity prediction approaches*

The first automated solution for assigning severity to bug reports was introduced by Menzies and Marcus (2008). Their solution, called SEVERity Issue Assessment (SEVERIS), provides fine-grained severity levels similar to NASA's for prioritization. They employed a machine learning classifier, utilizing feature vectors composed of top $k$ feature words, to forecast bug report severity. Haering et al. (2021), Lamkanfi et al. (2011) extended the work of Haering et al. (2021), Menzies and Marcus (2008) by applying their approach to open-source bug repositories, specifically analyzing textual descriptions of bug reports from prominent projects like GNOME, Eclipse, and Mozilla. To simplify the classification process, they categorized bug reports into severe and nonsevere groups, reducing severity labels from six to five.

Alenezi and Banitaan (2013) presented a priority prediction solution employing naive Bayes, decision trees, and random forests. Two distinct feature sets were employed: the first was based on term frequency weighted words extracted from bug reports, while the second incorporated operating system and severity classification information. Their investigation revealed that the second feature set performed better than the first. Random forests and decision trees outperformed naive Bayes in predictions. Gujral et al. (2015) specifically focused on classifying Eclipse bug reports and achieved 72% precision and 69% accuracy in their classification. They also introduced an algorithm for creating dictionary terms to predict severity levels by selecting a specific component. Zhang et al. (2015) explored the effectiveness of various classification algorithms in predicting bug report severity. They employed multiple machine-learning algorithms on a dataset of 29,204 bug reports. They found that naive Bayes multinomial outperformed conventional naive Bayes, one (1) nearest neighbor, and SVM.

Tian et al. (2015) used the nearest neighbor approach to predict bug severity. They consulted a large collection of over 65,000 Bugzilla reports. In an independent study, Choudhary (2017) proposed a bug severity level model that assigned priorities to Firefox crash reports on the Mozilla Socorro server. The model leveraged SVM to consider crash frequency and entropy. Kumari et al. (2018) presented an approach to assessing the severity of bug reports by incorporating entropy as a metric for measuring uncertainty and irregularities in the data. They evaluated various classifiers and validated their approach using PITS, Eclipse, and Mozilla projects, demonstrating their superiority over existing research. In another research work, Yang et al. (2017) developed an emotion words-based dictionary for analyzing textual emotions expressed in bug reports. They modified the naive Bayes multinomial algorithm and introduced the EWD multinomial algorithm. This outperformed previous severity prediction approaches (Khalajzadeh et al., 2022; Lamkanfi et al., 2010, 2011; Izadi et al., 2022).

### *2.2. Deep learning based severity prediction approaches*

Recent research (Oliaee et al., 2023; Xu et al., 2022; Gomes et al., 2022; Saga et al., 2022) has witnessed a surge of interest in deep learning techniques across various domains, such as computer vision, speech recognition, and sentiment analysis, due to their remarkable achievements (Graves et al., 2013; Ouyang et al., 2015; Sharma et al., 2012; Khalajzadeh et al., 2022). However, deep learning has not been extensively explored for predicting bug severity in bug reports. Nonetheless, there have been notable contributions in this area, which are discussed below.

Yu et al. (2010) introduced a framework based on artificial neural networks to predict bug report priorities in an international healthcare company. This framework, evaluated through threefold cross-validation experiments on five different products, demonstrated improvements in precision, recall, and F1-score. Sharma et al. (2012), Abi Kanaan et al. (2023) developed a priority prediction approach utilizing support vector machines, naive Bayes, k-nearest neighbors, and neural networks. Their goal was to predict the priority of newly arrived bug reports. The study indicated that different machine-learning techniques could identify the priority of fewer than 70% of bugs in Eclipse and OpenOffice projects (Gomes et al., 2023; Yu et al., 2023).

Ramay et al. (2019) proposed an automatic approach to severity prediction of bug reports using a deep neural network. They employed natural language processing techniques for text preprocessing, computed emotion scores for each bug report, and utilized a convolutional neural network for severity prediction. They mentioned a number of available tools to compute the sentiment of the written general text or software engineering text, i.e., SentiWordNet (Uddin et al., 2016), SentiStrengthSE (Islam and Zibran, 2018b), SentiCR (Ahmed et al., 2017a), Senti4SD (Calefato et al., 2018b) and EmoTxt (Calefato et al., 2017a). However, they found Senti4SD as best for the severity prediction of the software bugs. Their approach achieved a significant improvement in the f-measure by 7.90%, establishing it as the state-of-the-art algorithm for predicting bug report severity and serving as the baseline for our approach.

To summarize, researchers have proposed diverse machine learning and deep learning approaches for predicting bug report severity. While most of these approaches focus on traditional machine/deep learning algorithms, they have not considered mobile apps' bug reports. Additionally, these approaches do not consider implicit data training for feature modeling. These distinguishing characteristics set BERT-SBR apart from existing approaches.

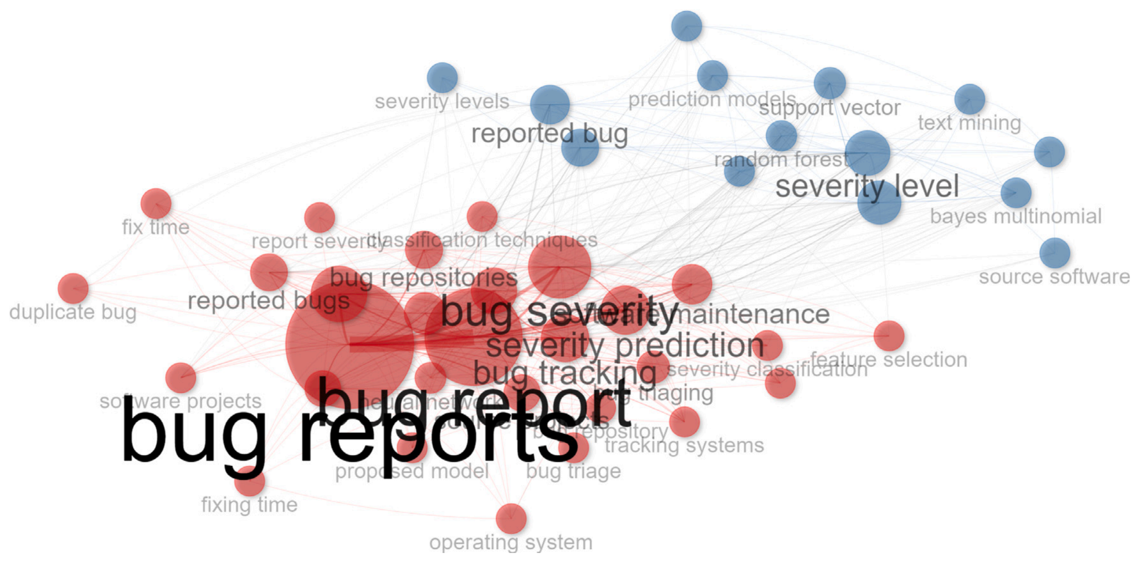


**Fig. 1.** Co-occurrence network.

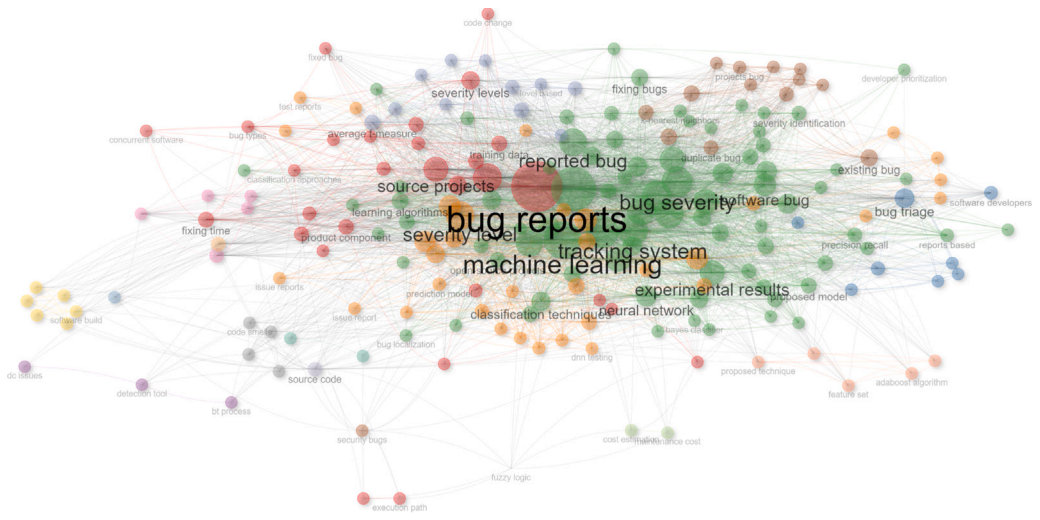


**Fig. 2.** Thematic map.

### 2.3. *Importance of the research*

To further check the significance of this topic, we performed a statistical analysis of the literature in R language. We extracted the *bibtxt* information from 121 most related papers using the *Web of Science* from 2008 to 2013. The collected papers are from 99 sources (58 journal papers, 62 proceeding papers, and 1 review paper) containing 2547 references and written by 382 authors. Given the mentioned data, we perform statistical analysis for the co-occurrence network, thematic map, and thematic evolution as shown in Fig. 1, Fig. 2, and Fig. 3, respectively.

Based on observations presented in Figs. 1–3, it is evident that the prediction of bug report severity in software has been scrutinized for several decades. This emphasizes the significance of bug reports and user feedback in software maintenance. However, it is worth noting that mobile apps were not initially considered part of software development. In the current landscape, with the increasing prevalence of mobile platforms, app maintenance has become increasingly crucial. It is imperative to recognize that software maintenance and app maintenance are distinct endeavors with unique characteristics. Therefore, accurate severity prediction for app bug reports becomes imperative in this context.

## 3. Approach

### 3.1. *Overview*

Fig. 4 illustrates an overview of BERT-SBR. The BERT-SBR performs the following steps for bug severity prediction in mobile apps.

- First, we collect the publicly available dataset of bug reports for mobile apps from the Hugging Face.[1]
- Then, the sentiment of the reporter from each bug report is computed using SentiWordNet3.4 repository.
- Next, we exploit the BERT for input formatting, i.e., padding, truncating, and attention masking.
- After that, we pass BERT formatted reviews and their sentiment to fine-tune the BERT classifier for severity prediction of bugs in mobile apps.
- Finally, the trained BERT classifier is tested to evaluate BERT-SBR.

[1] https://huggingface.co/datasets/app_reviews/blob/main/app_reviews.py

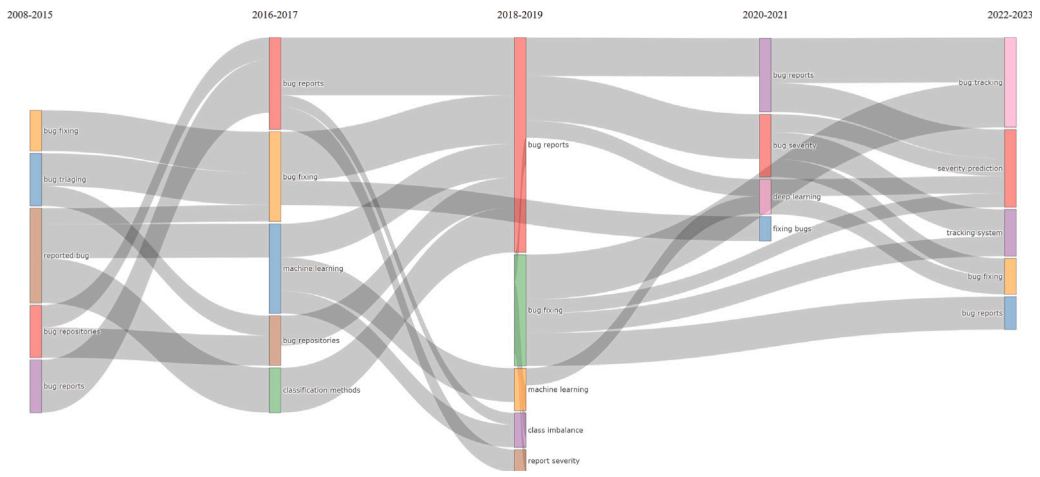


**Fig. 3.** Thematic evolution.

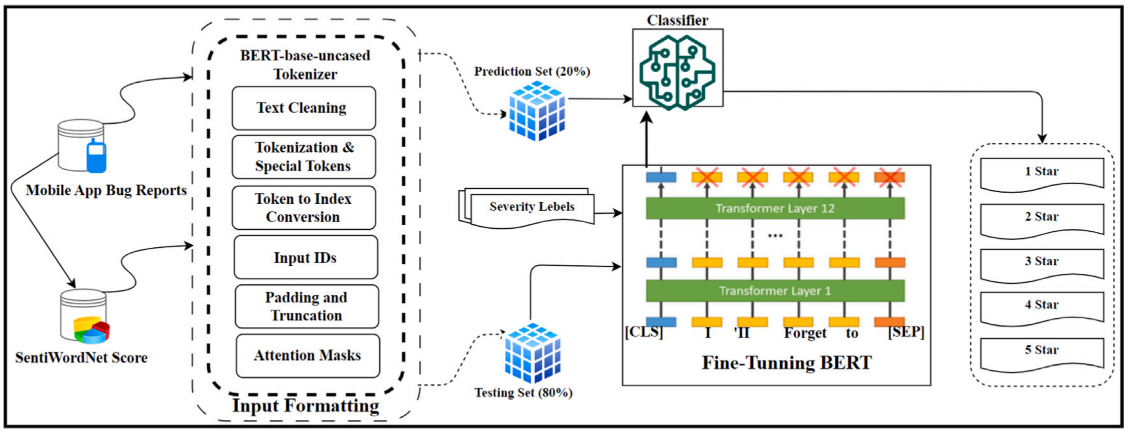


**Fig. 4.** BERT-SBR model.

### 3.2. Illustrating example

The given example is leveraged to explain the working of BERT-SBR for the severity prediction of bugs in mobile apps. It is a bug report of the package *com.mantz_it.rfanalyzer* collected from the extracted dataset from the Hugging Face.

- **Review** = “I’ll forgo the refund. But no go with Watson dongle... Nexus 9. Yet to try on Nexus 6p. So very disappointed!!! :-(”
- **Date** = “March 31 2016”
- **Severity** = “1”

The following sections explain the key steps of BERT-SBR.

### 3.3. Problem definition

The collected dataset contains the severity information of bugs in mobile apps and the related information. Given the set of bug reports for mobile apps $V$, a bug report $v$ can be written as follows:

$$v = \langle t, d, s \rangle \tag{1}$$

in which, $t$, $d$, and $s$ are the text of the bug report, date of review posting, and severity of $v$, respectively.

However, the illustrating example can be written as follows:

$$v' = \langle t', d', s' \rangle \tag{2}$$

in which, $t'$ = “I’ll forgo the refund. But no go with Watson dongle... Nexus 9. Yet to try on Nexus 6p. So very disappointed!!! :-(, $d'$ = date of review posting, and $s' = 1$

The BERT-SBR predicts the severity of a new bug in the mobile app from *1* to *5*. *1* indicates that the review represents an important and urgent bug. Whereas, *5* indicates that the posted review can be delayed as it does not contain critical issues related to the mobile apps. As a result, the severity prediction of bugs in a bug report $v$ can be defined as a mapping function as follows:

$$f : v \rightarrow c$$

$$c \in \{1,\ 2,\ 3,\ 4,\ or\ 5\}, \quad v \in \mathbb{V} \tag{3}$$

in which, $c$ is a predicted severity from *1* to *5*, $v$ is a mobile app review having bugs information, and $\mathbb{V}$ is a set of mobile app reviews.

### 3.4. SentiWordNet-based sentiment detection of bug reporters

The fundamental objective of sentiment analysis in Natural Language Processing (NLP) is to determine the sentiment tone conveyed by a text. It involves assessing whether the text of the bug report expresses a positive or negative opinion. BERT-SBR performs sentiment calculations on each individual bug report to classify sentiment accurately. Multiple sentiment analysis tools are available to assess sentiment from text documents, i.e., SentiWordNet (Baccianella et al., 2010). Moreover, there are several other tools specialized in software engineering for text classification. These tools include SentiStrengthSE (Islam and Zibran, 2018c), SentiCR (Ahmed et al., 2017b), Senti4SD (Calefato et al., 2018a), EmoTxt (Calefato et al., 2017b), and DEVA (Islam and Zibran,

**Table 1**
SentiWordNet-based sentiments of bug reporters.

| Review | Sentiment score | Severity | Sentiment |
|---|---|---|---|
| Fantastic application! It performs excellently on the Galaxy Note 5. | 0.65 | 4 | Positive |
| I will forgo the refund. But no go with the Watson dongle... Nexus 9. Yet to try on Nexus 6p. So very disappointed!!! :-( | 0.25 | 1 | Negative |
| The application functions smoothly with my Hackrf. Hopefully, there will be upcoming updates that bring additional features. | 0.5 | 5 | Neutral |

2018a). We chose SentiWordNet for two reasons: (1) SentiWordNet's extensive WordNet coverage enables effective sentiment analysis of diverse texts. In contrast, other tools with smaller or domain-specific lexicons may limit analyzing certain types of text or domains; (2) SentiWordNet is a versatile general-purpose sentiment analysis tool applicable to various domains, while tools, i.e., SentiCR, Senti4SD, EmoTxt, and DEVA are only specialized in software engineering sentiment analysis (Lin et al., 2018; Jongeling et al., 2017; Calefato et al., 2018a). Note that SentiWordNet outperforms the software engineering sentiment analysis tools, i.e., Senti4SD (mentioned in Section 5.9), in sentiment detection with the exploited dataset (Section 5.2). Therefore, this study recommends *SentiWordNet* for sentiment analysis to predict the severity of bug reports of mobile apps.

SentiWordNet is built on top of WordNet, which contains many words and their various senses or meanings. In WordNet, words can have multiple senses or meanings, and each sense is assigned a unique identifier called a "synset" (synonym set). For example, the word "bank" can have different connotations related to a financial institution and a riverbank. SentiWordNet assigns sentiment scores to each word sense in WordNet. These sentiment scores are represented as positive, negative, and neutral scores ranging from 0 to 1, which are shown in Table 1. The sentiment scores in SentiWordNet are derived from manually annotated data and based on the concept of a "bag of words". Each word sense is associated with a set of positive and negative terms, and their frequencies in a large text corpus are used to calculate the sentiment scores. When a word has multiple senses (synsets), SentiWordNet combines the sentiment scores of all the senses to calculate the overall sentiment score for the word. The aggregation process considers the frequency of each sense and the likelihood of it being used in a specific context. To analyze the sentiment of a sentence or text document, SentiWordNet breaks down the text into individual words. It looks up their sentiment scores in the SentiWordNet lexicon. It then combines the sentiment scores of all words in the sentence or document to obtain an overall sentiment score. To classify sentiment as positive, negative, or neutral, SentiWordNet often uses a threshold value. For example, if the overall sentiment score is greater than a certain threshold (e.g., 0.5), the text is classified as *positive*; if it is lower than the threshold, it is considered *negative*; and if it is equal to the threshold, it is considered *neutral*.

BERT-SBR passes the textual description $t$ of each bug report $v$ to the SentiWordNet that computes its sentiment. The sentiment of each bug report is combined with its text. A bug report with its sentiment can be written as follows:

$$v' = \langle t', d', s', n' \rangle \tag{4}$$

in which, $n'$ represents the sentiment of $v'$. Table 1 represents a few example bug reports and their sentiments.

### 3.5. BERT tokenizer-based input formatting

The severity prediction of a bug report is closely linked to how essential words are presented to BERT. To train BERT, data needs to be transformed into a specific format. Recent studies (Hu and Chen, 2016; Singh et al., 2017) explore various word representation methods in NLP, i.e., Word2Vec (Singh et al., 2017), GloVe (Hu et al., 2017), and FastText (Chen et al., 2018). However, BERT (Devlin et al., 2018) stands out as a potent NLP model pre-trained from an extensive text corpus. BERT has achieved outstanding performance on various Natural Language Understanding (NLU) tasks, i.e., text classification and question answering. Additionally, BERT offers several advantages, including quicker development, reduced data requirements, and improved outcomes (Devlin et al., 2018). Its remarkable capacity to learn contextually rich representations of words and phrases makes it highly effective for diverse tasks.

To convert data into a particular format, each bug report's text undergoes tokenization using the *uncased BERT tokenizer*. This process involves creating word tokens and adding a special token **[CLS]** at the beginning and another special token **[SEP]** at the end. For classification tasks, it is essential to include the **[CLS]** token at the beginning. The resulting tokens are then mapped to their corresponding indexes in the tokenizer vocabulary. To maintain uniformity, a sequence length is chosen (with a maximum of 402), and all reviews are either truncated or padded to achieve a fixed length. This study uses three different sequence lengths (128, 256, 402) for experimentation. Finally, attention masks distinguish between real and padded tokens.

The process of input formatting (presented in Table 2) outlines how data is prepared to be utilized with BERT for word embedding generation and classification. To achieve this, the "BertTokenizer" from the *Transformers* library, specifically "Bert-base-uncased", is employed. The formatting procedure for $v$ can be summarized as follows:

#### 3.5.1. Text cleaning

It is common to clean the text by removing unwanted characters, punctuation, or special symbols that might interfere with tokenization. BERT preprocesses data to ensure that the input text is clean and consistent.

#### 3.5.2. Word tokenization

The text is split into individual tokens or words, a process known as word splitting or word tokenization. The text of bug report is divided into tokens using the *BertTokenizer*, and these tokens are subsequently converted into integer IDs using a pre-defined vocabulary. In some cases, words (tokens) may be further broken down into subword units using techniques like byte-pair encoding (BPE) or WordPiece. For example, the word "unhappiness" might be split into "un" and "happiness" as subwords. The tokenization process can be written as follows:

$$S = [t1, t2, t3, \dots, tn] \tag{5}$$

in which, $S$ represents the sequence of tokens of $v$ and $t_i$ is a token, and $n$ is the total number of tokens per bug report ($\mathbf{v}$). Each token in the sequence corresponds to a specific part of the text of bug report.

Table 2
BERT input formatting for the example bug report.

| Step | Text | Output |
|---|---|---|
| Text Cleaning | I’ll forgo the refund. But no go with Watson dongle... Nexus 9. Yet to try on Nexus 6p. So very disappointed!!! :-( | I’ll forgo the refund. But no go with Watson dongle Nexus 9 Yet to try on Nexus 6p So very disappointed |
| Word Tokenization | I’ll forgo the refund. But no go with Watson dongle Nexus 9 Yet to try on Nexus 6p So very disappointed | [‘I’, “’”, ‘ll’, ‘forgo’, ‘the’, ‘refund’, ‘.’, ‘But’, ‘no’, ‘go’, ‘with’, ‘Watson’, ‘dongle’, ‘Nexus’, ‘9’, ‘Yet’, ‘to’, ‘try’, ‘on’, ‘Nexus’, ‘6’, ‘p’, ‘So’, ‘very’, ‘disappointed’] |
| Add Special Tokens | [‘I’, “’”, ‘ll’, ‘forgo’, ‘the’, ‘refund’, ‘.’, ‘But’, ‘no’, ‘go’, ‘with’, ‘Watson’, ‘dongle’, ‘Nexus’, ‘9’, ‘Yet’, ‘to’, ‘try’, ‘on’, ‘Nexus’, ‘6’, ‘p’, ‘So’, ‘very’, ‘disappointed’] | [‘[CLS]’, ‘I’, “’”, ‘ll’, ‘forgo’, ‘the’, ‘refund’, ‘.’, ‘But’, ‘no’, ‘go’, ‘with’, ‘Watson’, ‘dongle’, ‘Nexus’, ‘9’, ‘Yet’, ‘to’, ‘try’, ‘on’, ‘Nexus’, ‘6’, ‘p’, ‘So’, ‘very’, ‘disappointed’, ‘[SEP]’] |
| Token to Index Conversion | [‘[CLS]’, ‘I’, “’”, ‘ll’, ‘forgo’, ‘the’, ‘refund’, ‘.’, ‘But’, ‘no’, ‘go’, ‘with’, ‘Watson’, ‘dongle’, ‘Nexus’, ‘9’, ‘Yet’, ‘to’, ‘try’, ‘on’, ‘Nexus’, ‘6’, ‘p’, ‘So’, ‘very’, ‘disappointed’, ’[SEP]’] | [101, 1045, 1005, 2222, 19275, 1996, 20289, 1012, 2021, 2053, 2175, 2007, 12215, 11889, 13998, 1023, 2664, 2000, 3046, 2006, 13998, 1020, 1052, 1041, 2061, 26153, 100, 102] |
| Input IDs | [101, 1045, 1005, 2222, 19275, 1996, 20289, 1012, 2021, 2053, 2175, 2007, 12215, 11889, 13998, 1023, 2664, 2000, 3046, 2006, 13998, 1020, 1052, 1041, 2061, 26153, 100, 102] | 101 1045 1005 2222 19275 1996 20289 1012 2021 2053 2175 2007 12215 11889 13998 1023 2664 2000 3046 2006 13998 1020 1052 1041 2061 26153 100 102 |
| Padding and Truncation | 101 1045 1005 2222 19275 1996 20289 1012 2021 2053 2175 2007 12215 11889 13998 1023 2664 2000 3046 2006 13998 1020 1052 1041 2061 26153 100 102 | 101 1045 1005 2222 19275 1996 20289 1012 2021 2053 2175 2007 12215 11889 13998 1023 2664 2000 3046 2006 13998 1020 1052 1041 2061 26153 100 102 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| Attention Masks | 101 1045 1005 2222 19275 1996 20289 1012 2021 2053 2175 2007 12215 11889 13998 1023 2664 2000 3046 2006 13998 1020 1052 1041 2061 26153 100 102 | 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 |

### 3.5.3. Add special tokens

The BERT tokenizer adds special tokens to the token sequence. These tokens are used to convey specific information to the BERT model during training and inference. The two main special tokens are **[CLS]** and **[SEP]**. CLS stands for “classification” and is inserted at the beginning of the token sequence. It is used in tasks like text classification and sentence pair tasks to represent the entire input sequence’s classification, and SEP stands for “separation” and is inserted between two sentences or sequences in tasks involving multiple sentence pairs. It marks the end of one sentence and the beginning of the next.

The tokens sequence with special tokens can be defined as follows:

$$S = [CLS] + t_1 + t_2 + t_3 ... + t_n + n' + [SEP] \tag{6}$$

### 3.5.4. Token to index conversion

The BERT tokenizer has a pre-defined vocabulary containing all possible tokens that a deep-learning model can understand. Each unique token in the vocabulary is associated with a unique numerical identifier, called an “index”. Each token in the sequence $S$ is mapped to its corresponding index in the BERT tokenizer’s vocabulary. For example, the BERT tokenizer’s vocabulary contains the following tokens (assumed) with corresponding indexes:

$$\begin{aligned}
&[CLS] \rightarrow \text{index } 101 \\
&t_1 \rightarrow \text{index } 102 \\
&t_2 \rightarrow \text{index } 103 \\
&t_3 \rightarrow \text{index } 104 \\
&\vdots \\
&t_n \rightarrow \text{index } 105 \\
&[SEP] \rightarrow \text{index } 106
\end{aligned}$$

### 3.5.5. Input IDs

The list of token indexes is then converted into a list of input IDs. These input IDs represent the final input to the BERT model. Numerical IDs use a pre-defined vocabulary that can be written as:

$$I = [101, id_1, id_2, \dots, id_n, 102] \tag{7}$$

in which, $I$ represents the sequence of input ids of $v$ and $id_i$ is an input id, and $n$ is the total number of input ids per bug report $v$.

Table 3
Embedding of text illustrating example.

| Project details | BERT embeddings |
|---|---|
| Example Sentence | [0.0012, −0.0034, 0.0029, −0.0076, 0.0045, …] |

### 3.5.6. Padding and truncation

BERT pads sentences to a fixed length for efficient batch processing. During this step of input formatting, token IDs are adjusted to ensure a maximum sequence length of 402 (the number of words in the longest review). If the total number of token IDs for a particular input sequence, is shorter than 402 tokens, it is padded using a special token ‘0’ to reach the fixed length. On the other hand, if the sequence contains more than 402 tokens, the remaining tokens are truncated to fit within the specified limit.

$$I'_{1:402} = \begin{cases} I_{1:m}[PAD]^{402-m} & \text{if } m < 402 \\ I_{1:402} & \text{if } m > 402 \end{cases} \tag{8}$$

in which, $I'$ represents the final input IDs after padding and truncation for $v$.

### 3.5.7. Attention masks

Attention masks distinguish actual tokens from padding tokens within the input sequence. This differentiation is essential because the attention mechanism in the transformer architecture relies on these masks to prioritize the real tokens while ignoring the padding tokens. The attention mask is a binary mask with 0s for padded tokens and 1s for real tokens. For example, if the original sequence length is $n$, and we pad the sequence to a fixed length of $m$, the attention mask will be [1, 1, …, 1, 0, 0, …, 0], where the first $n$ elements are 1s (corresponding to real tokens), and the remaining $m$-$n$ elements are 0s (corresponding to padded tokens). The attention mask $MI'$ for input sequence $I$ can be written as:

$$M_{I'} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

**Table 4**
Hyperparameters for fine-tuning the proposed model.

| Data samples | MAX length | Batch size | Epochs | Learning rate | Epsilon | Attention heads | Hidden size | Input shape |
|---|---|---|---|---|---|---|---|---|
| **50000** | 402 | 16 | 10 | 2.10–5 | 1.00E–08 | 12 | 768 | [50000,402] |
| | 256 | 16 | 10 | 2.10–5 | 1.00E–08 | 12 | 768 | [50000,256] |
| | 128 | 8 | 10 | 2.10–5 | 1.00E–08 | 12 | 768 | [50000,128] |
| **5000** | 402 | 16 | 10 | 2.10–5 | 1.00E–08 | 12 | 768 | [5000,402] |
| | 256 | 16 | 10 | 2.10–5 | 1.00E–08 | 12 | 768 | [5000,256] |
| | 128 | 8 | 10 | 2.10–5 | 1.00E–08 | 12 | 768 | [5000,128] |

In the matrix representation, the row of 1's signifies the presence of actual tokens in the input sequence. The row of 0's represents padding tokens.

The pre-trained BERT model also generates embedding vectors that can be used to train other machine and deep learning models as shown in Table 3 for the illustrating example presented in Section 3.2.

## 4. Proposed model

Training a BERT model refers to the process of updating the model's parameters or weights using a large amount of labeled data. This is done to fine-tune it for a specific NLP task. BERT is a pre-trained language model that has already been exposed to vast amounts of text from various sources to learn general language representations (Devlin et al., 2018). During the pre-training phase, the model learns how to understand the meaning of words in a given sentence, as well as the semantic relationships between them. However, pre-training alone is insufficient for solving specific NLP tasks like text classification, named entity recognition, question-answering, etc. Therefore, by adding a task-specific layer, the BERT model needs to be further fine-tuned or trained on task-specific labeled data to adapt it to the particular task. A task-specific layer follows the pre-trained BERT layer.

The pre-trained BERT model can be adapted for specific tasks by fine-tuning its parameters using task-specific data, i.e., the mobile apps dataset. For the classification task, BERT-SBR fine-tunes the pre-trained BERT classifier and then utilizes the fine-tuned BERT to establish the correlation between app review features and severity status. BERT's superiority over other models stems from its capability to pre-train extensive text data, which enables it to grasp contextual relationships between words and sentences effectively. Due to this advantage, BERT outperforms other models in various NLP tasks. Additionally, BERT's attention mechanisms are well-suited to handling long text sequences.

### *4.1. Fine-tuning of the proposed model and prediction*

For fine-tuning and prediction, bug reports are randomly divided into two groups: 80% for training, and 20% for validation. The training dataset consists of 40,000 samples, while the validation dataset comprises 10,000 samples. The token IDs, attention masks, and labels of the training–validation dataset are merged into TensorDatasets, which are then used as input for the BERT model. To predict severity, BERT-SBR uses Google's English-specific BERT-based-uncased word-piece model,[2] This model is pre-trained on large text corpora, i.e., BookCorpus and Wikipedia, providing valuable contextual information for fine-tuning. The classification task utilizes the "BertForSequenceClassification" model, incorporating the hyperparameters recommended by the BERT authors (Devlin et al., 2018). Fine-tuning may involve adjusting various hyperparameters, i.e., learning rate, batch size, and number of training epochs, to find the optimal configuration that yields the most exceptional results. For BERT fine-tuning, batch sizes of 16 and 32 are recommended by the authors (Devlin et al., 2018). In BERT-SBR, the BERT base model, which consists of 12 transformer layers, 12 attention heads, and 768 hidden layers, is fine-tuned using Google Colab. The fine-tuning process involves using an epsilon value of 1e-8, a maximum sequence length of 402 token ids, running 10 epochs, and employing a learning rate of 2.10–5. Google Colab offers a complimentary GPU (either Tesla T4 or Tesla K80) based on availability. The hyperparameters employed to fine-tune the BERT base model in the BERT-SBR approach are detailed in Table 4. The classifier is trained and validated using a sequence length 402 and a batch size 16. During fine-tuning, the model passes through 10 training epochs, where it makes predictions on the input data, compares them with the true labels, and updates its parameters (weights) using optimization algorithms AdamW. The goal is to minimize prediction error and improve model performance on the specific task. After fine-tuning, BERT-SBR evaluates the trained model on a separate validation dataset to assess its performance on unseen data. This step ensures that the model generalizes well and is ready for deployment.

## 5. Evaluation

In this section, we begin by formulating research questions aimed at evaluating BERT-SBR. Next, we explore the process of gathering reviews and define the process of BERT-SBR. After that, we define the metrics and the evaluation procedure. Finally, we present the key findings and address potential threats to validity encountered during research question exploration.

### *5.1. Research questions*

The evaluation access BERT-SBR by investigating the following research questions:

- **RQ1:** What is the accuracy of BERT-SBR in predicting app review severity?
- **RQ2:** To what extent does re-sampling impact the performance of BERT-SBR?
- **RQ3:** How well does BERT-SBR perform compared to other machine learning algorithms in app review severity prediction?
- **RQ4:** What makes BERT feature extraction different from other methods (Word2Vec)?
- **RQ5:** What is the impact of SentiWordNet on BERT-SBR?

The first research question (RQ1) focuses on evaluating BERT-SBR accuracy. To compare our prediction algorithms, we use both random prediction and zero-rule algorithms. These algorithms are commonly employed when researchers lack a baseline approach for comparison, especially for rare or uncommon problems. By using distinct actual outcome values from the training data and predicting random outcome values from the testing data, the random prediction algorithm can forecast random outcome values from the testing data. Conversely, the zero-rule algorithm predicts the classification that occurs most frequently within a given dataset. To address the second research question (RQ2), we explore the influence of re-sampling, considering that our dataset is imbalanced, with a significant bias towards *5-stars* reviews. Re-sampling can be achieved through three methods, i.e., under-sampling, over-sampling, or adjusting the classifier threshold value. Under-sampling involves reducing data in an imbalanced

[2] https://github.com/google-research/bert accessed on 15 July 2023.

**Table 5**
Dataset statistics.

| App reviews | |
|---|---|
| Total No. of App Reviews | 50 000 |
| App Review Categories | 5 |
| **Star Rating** | **Count (Percentage)** |
| 1 Star | 7580 (15.16%) |
| 2 Star | 2995 (5.99%) |
| 3 Star | 4723 (9.446%) |
| 4 Star | 7625 (15.25%) |
| 5 Star | 27077 (54.154%) |
| **Words in app review** | |
| Minimum Words | 1 |
| Maximum Words | 422 |

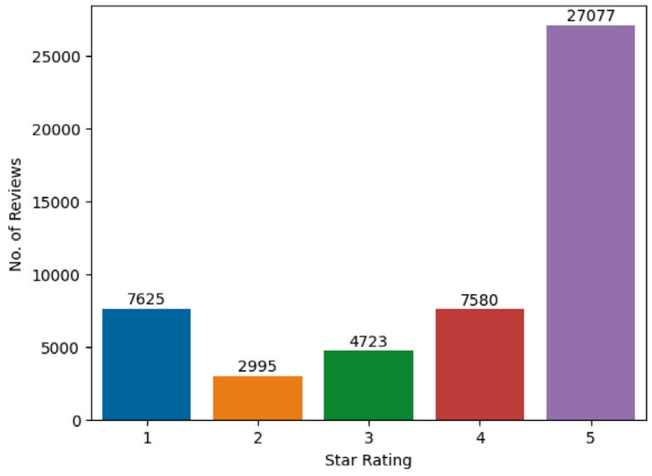


**Fig. 5.** Severity breakdown by class.

dataset to balance the classes, while over-sampling adds data to create balance. Alternatively, the classifier threshold value can be adjusted to assign weight to each class, achieving a balanced dataset. In order to investigate the effect of resampling on our findings, we utilize both oversampling and undersampling.

In the third research question (RQ3), BERT-SBR's performance in predicting app review severity is evaluated against other machine learning algorithms. This assessment aims to determine how effectively BERT-SBR outperforms or compares with alternative algorithms in predicting the severity of app reviews. The fourth research question (RQ4) focuses on evaluating the influence of BERT embeddings on BERT-SBR. To achieve this, we leveraged Word2Vec and BERT to generate word embeddings. Then, we compare their respective results by inputting these embeddings into the Recurrent Neural Network (RNN) model. The goal is to analyze and understand the impact of using BERT embeddings compared to Word2Vec embeddings when predicting review severity with the RNN model.

Finally, the last research question (RQ5) investigates the influence of reporters' sentiment to predict the severity of the app review. To ascertain the true impact of SentiWordNet on BERT-SBR, comprehensive experimentation, and evaluation are conducted, comparing the model's performance (accuracy, precision, recall, and F-measure) with and without SentiWordNet integration. This analysis allows us to gain insights into the potential benefits of leveraging SentiWordNet in sentiment-aware BERT-SBR models for app review severity prediction.

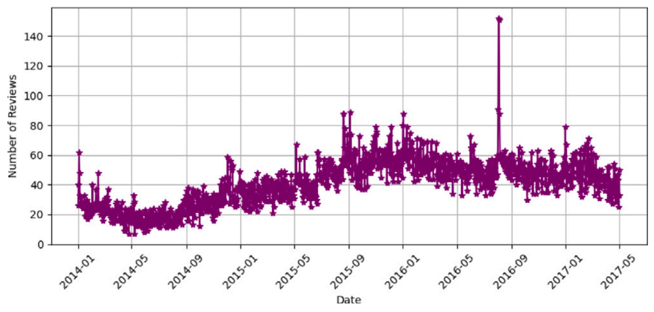


**Fig. 6.** Review count over time.

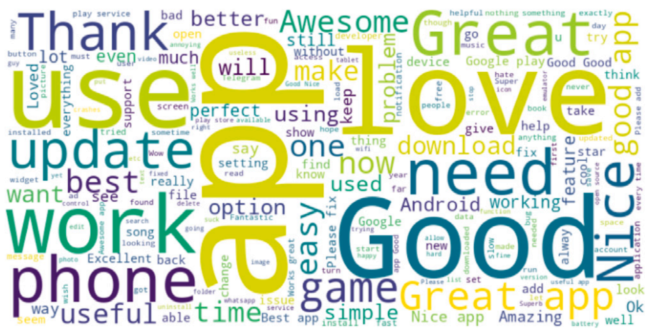


**Fig. 7.** Most common words in app reviews.

### 5.2. Dataset

App reviews historical dataset is acquired from *Hugging Face*[3] created by dataset curators,[4] leveraging the *datasets* library. The dataset is composed of monolingual English messages. The data extraction process includes retrieving various attributes, i.e., package name, review, date, and star (severity) until May 2017, as shown in Fig. 6. Table 5 presents an overview of the dataset statistics. The dataset encompasses a substantial collection of Android applications categorized into 23 distinct app categories. It offers a comprehensive view of the feedback provided by users for these apps and also documents the progression of the associated code metrics. The dataset includes approximately 395 applications sourced from the F-Droid repository, comprising around 600 different versions. It consists of a vast corpus of approximately 280,000 user reviews, selected 50,000 for BERT-SBR, extracted using specialized text-mining techniques. According to Table 5, the selected dataset consists of 50000 records, each representing a review's severity as "1 star", "2 star", "3 star", "4 star", or "5 star". Among the total reviews, 7625 (15.25%) are marked as "1 star", 2995 (5.99%) as "2 star", 4723 (9.446%) as "3 star", 7580 (15.16%) as "4 star", and 27077 (54.154%) as "5 star", as shown in Fig. 5. Additionally, the review details contain a minimum of 1 word and a maximum of 422 words. Fig. 7 gives a quick visual overview of the most frequently occurring words or terms within app reviews text. In the cloud, the more frequently occurring words appear larger and more prominent, while the less frequently occurring words appear smaller and less prominent.

[3] https://huggingface.co/datasets

[4] Giovanni Grano - (University of Zurich), Sebastiano Panichella - (University of Zurich), Andrea di Sorbo - (University of Sannio).

### 5.3. Evaluation metrics

To evaluate BERT-SBR performance, we utilize standard metrics commonly used in multiclassification classification, i.e., accuracy, precision, sensitivity (recall), and f-measure.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

$$Precision = \frac{TP}{TP + FP} \tag{10}$$

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

$$F - measure = \frac{2 \times Precision \times Reccall}{Precision + Recall} \tag{12}$$

in which, $TP$ represents the count of reviews correctly classified by their respective star rating (successful classification). $TN$ represents the count of reviews correctly classified as not belonging to the respective star rating class (unsuccessful classification). $FP$ denotes the count of reviews incorrectly classified as belonging to a star rating class (false positive). $FN$ indicates the count of reviews incorrectly classified as not belonging to a star rating class (false negative).

### 5.4. Process

To evaluate BERT-SBR, the following steps are followed. Firstly, we extracted app reviews $V$ from Hugging Face. Subsequently, the data underwent automatic preprocessing by the pre-trained BERT tokenizer. This preprocessing involved tasks, i.e., sentence tokenization, punctuation removal, converting all characters to lowercase, stop words removal, adding special characters, mapping to token IDs, and fixing the length by padding and truncation. Next, a hold-out validation technique is applied to $V$, wherein the data is split into an 80%–20% ratio. Significantly, we utilize 80% of sorted app reviews for our training set, reserving 20% of recent app reviews for testing. We opt for hold-out validation over k-fold cross-validation because we aim to harness historical data for new projects. Finally, the pre-trained BERT model (*BertForSequenceClassification*) is fine-tuned using the training samples, and the performance of the fine-tuned BERT model is evaluated on the testing samples.

To validate BERT-SBR, the process involves the following steps:

- Initially, a set of training reviews is selected.
- Next, the training samples are utilized to fine-tune the pre-trained BERT.
- The training samples are used to train several models, including RNN, Long Short-Term Memory (LSTM), Neural Network (NN), Convolutional Neural Network (CNN), Logistic Regression (LR), Support Vector Machine (SVM), Random Forest (RF), Adaptive Boosting (AdaBoost), and Decision Tree (DT).
- Fourthly, using the trained BERT, RNN, LSTM, NN, CNN, LR, SVM, Random Forest, AdaBoost, and DT models, we predict the severity status of each review from the testing samples, respectively.
- Our final step is to analyze each classifier's accuracy, precision, recall, and f-measure to measure how well it performs.

### 5.5. Evaluation results

#### 5.5.1. Accuracy of BERT-SBR

To address the RQ1, we compare the developed approach and two baseline algorithms, namely random prediction and zero rule. These baseline algorithms serve as benchmarks to assess the BERT-SBR's accuracy. THE BERT-SBR is the first one to our knowledge for predicting review severity since there are no existing approaches for performance comparison. Hence, we selected these algorithms to evaluate and compare the BERT-SBR's performance.

**Table 6**
Comparison of BERT-SBR against baseline approaches (%).

| Model | Accuracy | Precision | Recall | F-measure |
|---|---|---|---|---|
| **BERT-SBR** | **91.13** | **91.02** | **91.13** | **91.03** |
| Zero Rule | 54.15 | 29.33 | 54.15 | 38.05 |
| Random Prediction | 27.11 | 29.34 | 27.11 | 28.18 |

Table 6 presents the evaluation results of BERT-SBR, random prediction, and zero rule approach. Columns 2–5 display the performance results of precision, recall, and f-measure for each classifier, respectively. Each row represents the performance of a specific classifier. The hold-out validation results, depicted in Table 6, showcase a comparison of various classifiers' performance. The findings demonstrate that BERT-SBR has an accuracy of 91.13%, precision of 91.02%, recall of 91.13%, and an f-measure of 91.03%, respectively. In comparison, the embedding-based zero rule approach achieves an accuracy of 54.15%, precision of 29.33%, recall of 54.15%, and an f-measure of 38.05%. Similarly, the embedding-based random prediction model achieves an accuracy of 27.11%, precision of 29.34%, recall of 27.11%, and an f-measure of 28.18%.

#### 5.5.2. Observations

Referring to Table 6, the following observations can be made:

- The BERT-SBR performs better than random prediction and zero rule classifiers.
- In terms of accuracy, precision, recall, and f-measure, the BERT-SBR improvement over both zero rule and random prediction is ((91.13%–7.11%) / 91.13% * 100% = 70.29% and (91.13%–4.15%) / 91.13% * 100% = 40.43%), ((91.02%–9.33%) / 91.02% * 100% = 67.78% and (91.02%–9.34%) / 91.02% * 100% = 67.77%), ((91.13%–4.15%) / 91.13% * 100% = 40.71% and (91.137%–7.11%) / 91.13% * 100% = 70.29%), and ((91.03%–8.05%) / 91.03% * 100% = 58.14% and (91.03%–28.18%) / 91.03% * 100% = 68.96%), respectively. One potential reason is that BERT's advanced capability to comprehend intricate word relationships makes it a potent tool in text classification.
- BERT outperforms the zero rule and random prediction classifiers due to its advanced language modeling capabilities and data-driven learning approach. With contextualized embeddings, BERT captures nuanced meanings in text and comprehends complex language patterns. Its deep learning architecture enables effective learning of intricate patterns and feature extraction. Furthermore, BERT's pre-training on a vast text corpus enhances language understanding, leading to superior performance during fine-tuning on specific tasks. This combination of features makes BERT highly effective in text classification, surpassing the simplicity and limitations of the zero rule and random prediction classifiers.

### 5.6. Influence of resampling

To investigate RQ2, we applies two re-sampling techniques to tackle the class imbalance in the dataset, i.e., over-sampling and under-sampling. Over-sampling involves generating additional samples for the minority class using RandomOverSampler (Fig. 8) while under-sampling addresses imbalanced datasets by removing excess majority class records using RandomUnderSampler (Fig. 9). As shown in Table 7, both resampled and not resampled evaluation results are presented. A resampling setting is shown in the first column, while accuracy, precision, recall, and f-measure results are shown in columns 2–5.

The findings demonstrate that applying the over-sampling technique yields notable improvements in accuracy, precision, recall, and f-measure, achieving values of 95.07%, 95.10%, 95.07%, and 95.06%, respectively. Likewise, the under-sampling technique results in accuracy, precision, recall, and f-measure values of 87.31%, 87.35%, 87.31%, and 87.32%, respectively.

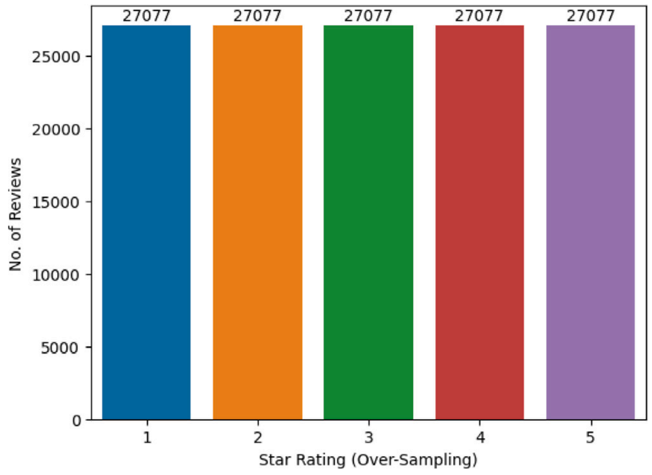


**Fig. 8.** Review count by class with over-sampling.

**Table 7**
Influence of re-sampling (in %).

| Re-sampling | Accuracy | Precision | Recall | F-measure |
|---|---|---|---|---|
| No (Default) | 91.13 | 91.02 | 91.13 | 91.03 |
| Yes (Under-Sampling) | 87.31 | 87.35 | 87.31 | 87.32 |
| Yes (Over-Sampling) | 95.07 | 95.1 | 95.07 | 95.06 |

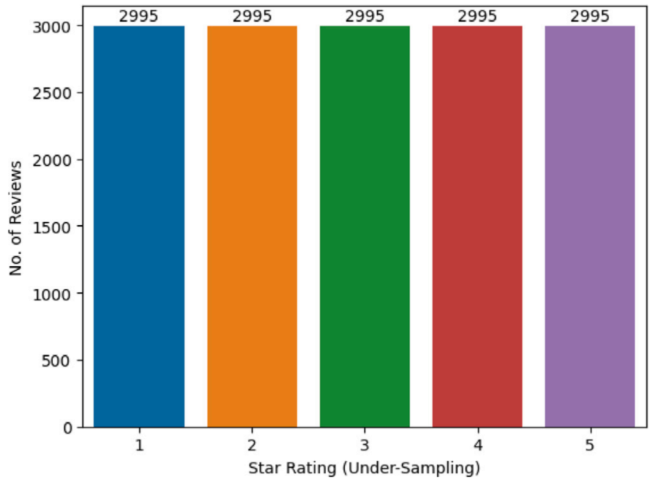


**Fig. 9.** Review count by class with under-sampling.

### 5.6.1. Observations

From Table 7, we make the following observations:

- Implementing the over-sampling technique results in notable enhancements in accuracy, precision, recall, and f-measure. Accuracy increased by 4.33%, precision by 4.49%, recall by 4.33%, and f-measure by 4.41%. These improvements can be attributed to BERT's exposure to a larger and more balanced dataset, enabling it to learn meaningful patterns more effectively. The additional data provided by the over-sampling technique has proven beneficial in enhancing the model's performance.
- Conversely, under-sampling involves reducing the number of samples in the majority class, leading to valuable information loss. Consequently, the majority and minority classes in the fine-tuned BERT model demonstrate lower performance when under-sampling is applied.

**Table 8**
Comparison of different classifiers (%) Based on BERT embeddings.

| Model name | Accuracy | Precision | Recall | F-measure |
|---|---|---|---|---|
| **BERT-SBR** | **91.13** | **91.02** | **91.13** | **91.03** |
| NB | 67.52 | 92.71 | 68.45 | 78.75 |
| MNB | 67.76 | 92.64 | 68.79 | 78.95 |
| BNB | 66.71 | 92.87 | 67.31 | 78.05 |
| RNN | 58.54 | 36.37 | 31.15 | 30.02 |
| LSTM | 58.28 | 33.97 | 31.22 | 29.17 |
| NN | 57.88 | 39.70 | 28.93 | 28.27 |
| CNN | 56.68 | 39.02 | 28.91 | 28.21 |
| LR | 56.94 | 52.24 | 56.94 | 47.07 |
| SVM | 55.10 | 39.14 | 55.10 | 40.93 |
| RF | 54.82 | 44.66 | 54.82 | 43.02 |
| AdaBoost | 53.93 | 28.56 | 24.31 | 21.76 |
| DT | 41.26 | 25.88 | 25.97 | 25.92 |

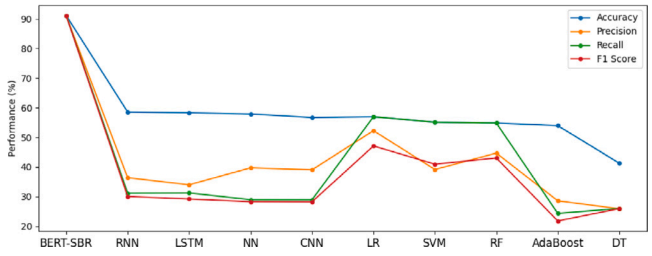


**Fig. 10.** Comparison of different classifiers (in %) based on BERT embeddings.

### 5.7. Performance comparison of machine learning algorithms

To address RQ3, we explore widely used text classification algorithms, i.e., RNN, LSTM, NN, CNN, Naive Bayes (NB), Multinomial NB, Bernaulli NB, LR, SVM, Random forest, AdaBoost, and DT, with BERT embeddings. We then compare their results with a fine-tuned BERT classifier. Parameter tuning, involving the optimization of statistical model fitting, is conducted to fine-tune the parameters of the mentioned classification algorithms. We train and evaluate these algorithms using various parameter configurations to identify optimal settings. Subsequently, we compare the performance of the classification algorithms based on the most optimal parameter settings.

The evaluation results of the BERT, RNN, LSTM, NN, CNN, LR, SVM, Random forest, AdaBoost, and DT classifiers are depicted in Table 8. Each row corresponds to a specific classifier, while columns 2–5 display accuracy, precision, recall, and f-measure, respectively. The BERT-SBR achieves impressive results with an accuracy of 91.13%, precision of 91.02%, recall of 91.13%, and f-measure of 91.03%, in contrast to other classifiers. The RNN achieves an accuracy of 58.54%, precision of 36.37%, recall of 31.15%, and f-measure of 30.02%. The MNB, NB, and BNB have accuracy of (67.76%, 67.52%, and 66.71%), precision of (92.64%, 92.71%, and 92.87%), recall of (68.79%, 68.45%, and 67.31%), and f-measure of (78.95%, 78.75%, 78.05%), respectively. LSTM has an accuracy of 58.28%, precision of 33.97%, recall of 31.22%, and f-measure of 29.17%. The NN classifier achieves an accuracy of 57.88%, precision of 39.7%, recall of 28.93%, and f-measure of 28.27%. Similarly, the CNN classifier obtains an accuracy of 56.68%, precision of 39.02%, recall of 28.91%, and f-measure of 28.21%. The LR classifier shows an accuracy of 56.94%, precision of 52.24%, recall of 56.94%, and f-measure of 47.07%. SVM achieves an accuracy of 55.1%, precision of 39.14%, recall of 55.1%, and f-measure of 40.93%. RF exhibits an accuracy of 54.82%, precision of 44.66%, recall of 54.82%, and f-measure of 43.02%. AdaBoost performs with an accuracy of 53.93%, precision of 28.56%, recall of 24.31%, and f-measure of 21.76%. DT obtains an accuracy of 41.26%, precision of 25.88%, recall of 25.97%, and f-measure of 25.92%.

**Table 9**
Influence of BERT embeddings (in %).

| Feature modeling | Accuracy | Precision | Recall | F-measure |
|---|---|---|---|---|
| With BERT (RNN) | 58.54 | 36.37 | 31.15 | 30.02 |
| With Word2Vec (RNN) | 53.55 | 32.77 | 31.11 | 29.02 |

**Table 10**
Influence of SentiWordNet (in %).

| Sentiment computation | Accuracy | Precision | Recall | F-measure |
|---|---|---|---|---|
| With SentiWordNet | 91.13 | 91.02 | 91.13 | 91.03 |
| With Senti4SD | 91.07 | 90.99 | 91.04 | 91.01 |
| Without SentiWordNet | 91.05 | 90.94 | 91.05 | 90.94 |

#### *5.7.1. Observations*

Based on the data presented in Table 8 and Fig. 10, we derive the following observations:

- BERT performs superior to RNN, LSTM, NN, CNN, LR, SVM, Random Forest, AdaBoost, and DT classifiers. The advantage is attributed to its extensive pre-training on a great deal of data, as a result of which it can develop highly generalized and meaningful representations of text. Moreover, BERT leverages attention mechanisms to capture long-range dependencies and contextual information, enhancing its proficiency in natural language understanding and processing tasks.
- Deep learning models (i.e., BERT, RNN, LSTM, NN, and CNN) perform better than machine learning models (i.e., LR, SVM, Random Forest, AdaBoost, and DT). This improvement can be attributed to deep learning models' ability to learn intricate and hierarchical data representations. This results in enhanced performance for tasks involving natural language understanding and processing. Additionally, deep learning models can efficiently handle large datasets and autonomously learn relevant features, eliminating manual feature engineering. This further contributes to their superior performance.
- LR exhibits superior performance to other machine and deep learning models in terms of precision, recall, and f-measure. This is due to its ability to handle linear relationships effectively and its robustness to imbalanced class distributions. However, its performance superiority is highly context-dependent and should be evaluated based on specific data and tasks.
- Although NB, MNB, and BNB have slight improvement (1.86%, 1.78%, and 2.03%) in precision, the proposed classifier significantly outperforms them in accuracy (37.97%, 34.49%, and 36.61%), recall (33.13%, 32.48%, and 35.39%), and f-measure (15.59%, 15.30%, and 16.63%). Higher precision and lower recall indicate that they are not fit for the imbalanced dataset in contrast to the proposed classifier (mentioned in Section 5.6) and result in many false positives and false negatives.

### *5.8. Influence of BERT embeddings*

In response to the fourth research question (RQ4), we compare BERT embeddings and Word2Vec embeddings on BERT-SBR using the RNN model, as presented in Table 9. The first column of the table indicates BERT-SBR performance with BERT and Word2Vec embeddings, while columns 2–5 display accuracy, precision, recall, and f-measure metrics. Each row corresponds to different BERT-SBR embedding results. With BERT embeddings, RNN achieves an accuracy of 58.54%, precision of 36.37%, recall of 31.15%, and f-measure of 30.02%. In contrast, with Word2Vec embeddings, RNN demonstrates an accuracy of 53.55%, precision of 32.77%, recall of 31.11%, and f-measure of 29.02%.

#### *5.8.1. Observations*

Referring to Table 9, the following observations can be made:

- By employing word embeddings generated from a pre-trained BERT model, the RNN model exhibits improvements of 9.30% in accuracy, 10.98% in precision, 0.13% in recall, and 3.45% in f-measure. These enhancements signify the positive impact of BERT embeddings on the RNN model's overall effectiveness in various performance metrics. BERT outperforms Word2Vec due to its superior architecture and capabilities. While Word2Vec is a shallow neural network model that predicts word context based on neighboring words. BERT is a transformer-based deep neural network model pre-trained on extensive data, enabling it to capture intricate relationships and dependencies between words in a sentence. Moreover, BERT's bidirectional nature allows it to consider context before and after a word in a sentence. In contrast, Word2Vec can only consider the preceding words. As a result, BERT-SBR (BERT embeddings) achieves higher review severity prediction performance than BERT-SBR (Word2Vec embeddings).

### *5.9. Influence of sentiwordnet*

To address the last research question (RQ5), we evaluate BERT-SBR with and without SentiWordNet. The evaluation outcomes are presented in Table 10, where the first column denotes the input settings, and columns 2–5 represent the accuracy, precision, recall, and f-measure performance metrics. On average, BERT-SBR with SentiWordNet and Senti4SD achieved accuracy, precision, recall, and f-measure values of (91.13%, 91.02%, 91.13%, and 91.03%) and (91.07%, 90.99%, 91.04%, and 91.01%), respectively. In contrast, BERT-SBR without SentiWordNet obtained average, precision, recall, and f-measure values of 91.05%, 90.94%, 91.05%, and 90.94%, respectively.

#### *5.9.1. Observations*

Referring to Table 10, the following observations can be made:

- The results indicate that incorporating SentiWordNet in the analysis using BERT-SBR leads to slight improvements in the performance metrics compared to using BERT-SBR without SentiWordNet. SentiWordNet improves average accuracy, precision, recall, and f-measure by 0.09%, 0.08%, 0.09%, and 0.10%, respectively. SentiWordNet's inclusion allows the BERT-SBR model to better understand the sentiment and context of words, resulting in a slightly enhanced ability to perform sentiment analysis and polarity classification tasks. Although the improvements are relatively small, they highlight the potential benefits of external lexical resources like SentiWordNet in natural language processing tasks. Note that SentiWordNet could be replaced with Senti4SD if one wants to consider a software engineering sentiment computation repository. Senti4SD also improves average accuracy, precision, recall, and f-measure by 0.07%, 0.03%, 0.09%, and 0.02%, respectively.

Based on the preceding analysis, it can be inferred that BERT-SBR's performance is significant in predicting the severity of bug reports in mobile apps.

### *5.10. Threats to validity*

One potential threat to construct validity is related to the choice of evaluation metrics. The selected metrics, including accuracy, precision, recall, and f-measure, are widely adopted in similar studies.

Consequently, we also utilize these metrics to assess the performance of BERT-SBR.

Another construct validity concern arises from the utilization of *SentiWordNet* for computing the sentiment of bug reports of mobile apps. We chose *SentiWordNet* based on its compatibility with our dataset. However, using different sentiment calculation repositories could potentially impact BERT-SBR's performance.

For internal validity, a threat stems from the implementation of the baseline approach. We took steps to validate the implementation and results, but there may still be undiscovered errors.

Regarding external validity, there are two main threats. Firstly, the generalizability of our results might be limited as we primarily focused on bug reports from mobile apps to evaluate BERT-SBR. Therefore, extending the applicability of BERT-SBR to bug reports from other projects may yield different outcomes.

Additionally, the performance of BERT-SBR could be affected by the hyper-parameter settings for deep learning algorithms. Variables, i.e., the size of the bug report training set, or the adjustment of hyper-parameters may influence the model's overall performance.

## 6. Conclusions and future work

Automated classification of bug reports of mobile apps is highly desirable for their maintenance. The severity level plays a pivotal role in prioritizing bug resolution, enabling developers to address critical bugs promptly. Nonetheless, manually assessing the severity of each issue can be laborious and prone to errors. In this paper, we propose a BERT-based severity prediction of bug reports. It leverages a deep neural network for automatic bug severity classification to maintain mobile apps. BERT-SBR computes the sentiment of reporters of the bug reports and preprocesses them by leveraging BertTokenizer input formatting techniques, passes the formatted text and computes the sentiment of each bug report to generate the word embeddings, introduces a fine-tuned BERT classifier for the severity prediction of bug reports, passes the generated word embeddings to the fine-tuned BERT classifier for the training and testing of the classifier. On average, BERT-SBR achieves a remarkable improvement of 40.43%, 67.78%, 40.71%, and 58.14% in accuracy, precision, recall, and f-measure, indicating its superiority in accurately predicting the severity of bug reports for the maintenance of mobile applications.

## CRediT authorship contribution statement

**Asif Ali:** Methodology, Data analysis and preprocessing, Paper writing. **Yuanqing Xia:** Conceptualization, Methodology, Paper revision, Response to reviewers. **Qasim Umer:** Algorithm implementation and evaluation, Paper revision, Response to reviewers. **Mohamed Osman:** Conceptualization, Paper revision.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

Data will be made available on request.

## Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grants 61836001, 62173035, and 62102022.

## References

Abi Kanaan, M., Couchot, J.-F., Guyeux, C., Laiymani, D., Atechian, T., Darazi, R., 2023. A methodology for emergency calls severity prediction: From pre-processing to BERT-based classifiers. In: IFIP International Conference on Artificial Intelligence Applications and Innovations. Springer, pp. 329–342.

Ahmed, T., Bosu, A., Iqbal, A., Rahimi, S., 2017a. Senticr: A customized sentiment analysis tool for code review interactions. In: Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering. In: ASE 2017, IEEE Press, Piscataway, NJ, USA, pp. 106–111, URL http://dl.acm.org/citation.cfm?id=3155562.3155579.

Ahmed, I., Rahman, M.M., Roy, C.K., 2017b. Senticr: A customized sentiment analysis tool for code review interactions. In: Proceedings of the 25th International Conference on Program Comprehension. pp. 312–315.

Alenezi, M., Banitaan, S., 2013. Bug reports prioritization: Which features and classifier to use? In: 2013 12th International Conference on Machine Learning and Applications, Vol. 2. pp. 112–116. http://dx.doi.org/10.1109/ICMLA.2013.114.

Atlassian, 2018. JIRA issue tracker. https://www.atlassian.com/software/jira/, Nov, 2018.

Baccianella, S., Esuli, A., Sebastiani, F., 2010. SentiWordNet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In: LREC 2010 - 7th International Conference on Language Resources and Evaluation. pp. 2200–2204.

Calefato, F., Lanubile, F., Maiorano, F., Novielli, N., 2018a. Senti4SD: A toolkit for sentiment analysis in software development. In: 2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice. (ICSE-SEIP), pp. 85–88.

Calefato, F., Lanubile, F., Maiorano, F., Novielli, N., 2018b. Sentiment polarity detection for software development. Empir. Softw. Eng. 23 (3), 1352–1382. http://dx.doi.org/10.1007/s10664-017-9546-9.

Calefato, F., Lanubile, F., Novielli, N., 2017a. EmoTxt: A toolkit for emotion recognition from text. In: 2017 Seventh International Conference on Affective Computing and Intelligent Interaction Workshops and Demos. (ACIIW), pp. 79–80.

Calefato, F., Novielli, N., Carrozza, C., Lanubile, F., Grano, G., Maiorano, F., 2017b. EmoTxt: A toolkit for emotion recognition from text. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice. (ICSE-SEIP), pp. 193–196.

Chaturvedi, K., Singh, V., 2012. Determining bug severity using machine learning techniques. In: Software Engineering (CONSEG), 2012 CSI Sixth International Conference on. IEEE, pp. 1–6.

Chen, C., Yang, Y., Zhou, J., Li, X., Bao, F., 2018. Cross-domain review helpfulness prediction based on convolutional neural networks with auxiliary domain discriminators. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Vol. 2. 2, pp. 602–607.

Choudhary, P., 2017. Neural network based bug priority prediction model using text classification techniques. Int. J. Adv. Res. Comput. Sci. 8 (5), 1315–1319. http://dx.doi.org/10.26483/ijarcs.v8i5.3559, URL http://www.ijarcs.info/index.php/Ijarcs/article/view/3559.

Devlin, J., Chang, M.-W., Lee, K., Toutanova, K., 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Gomes, L., Côrtes, M., Torres, R., 2022. Bert-based feature extraction for long-lived bug prediction in floss: A comparative study. Available at SSRN 4166555.

Gomes, L., da Silva Torres, R., Côrtes, M.L., 2023. BERT-and TF-IDF-based feature extraction for long-lived bug prediction in FLOSS: A comparative study. Inf. Softw. Technol. 160, 107217.

Graves, A., Mohamed, A., Hinton, G.E., 2013. Speech recognition with deep recurrent neural networks. arXiv:1303.5778, URL http://arxiv.org/abs/1303.5778.

Gujral, S., Sharma, G., Sharma, S., 2015. Classifying bug severity using dictionary based approach. In: 2015 1st International Conference on Futuristic Trends in Computational Analysis and Knowledge Management. ABLAZE 2015, http://dx.doi.org/10.1109/ABLAZE.2015.7154933.

Haering, M., Stanik, C., Maalej, W., 2021. Automatically matching bug reports with related app reviews. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering. (ICSE), IEEE, pp. 970–981.

Hu, Y.-H., Chen, K., 2016. Predicting hotel review helpfulness: The impact of review visibility and interaction between hotel stars and review ratings. Int. J. Inf. Manage. 36 (6), 929–944.

Hu, Y.-H., Chen, K., Lee, P.-J., 2017. The effect of user-controllable filters on the prediction of online hotel reviews. Inf. Manag. 54 (6), 728–744.

Islam, M.S., Zibran, M.F., 2018a. DEVA: A sentiment analysis tool for software developers. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. pp. 267–268.

Islam, M.R., Zibran, M.F., 2018b. SentiStrength-SE: Exploiting domain specificity for improved sentiment analysis in software engineering text. J. Syst. Softw. 145, 125–146. http://dx.doi.org/10.1016/j.jss.2018.08.030, URL http://www.sciencedirect.com/science/article/pii/S0164121218301675.

Islam, M.S., Zibran, M.F., 2018c. SentiStrengthSE: A sentiment analysis tool for software engineering text. In: 2018 IEEE/ACM 40th International Conference on Software Engineering. (ICSE), pp. 970–973.

Izadi, M., Akbari, K., Heydarnoori, A., 2022. Predicting the objective and priority of issue reports in software repositories. Empir. Softw. Eng. 27 (2), 50.

Jongeling, R., Bacchelli, A., van Deursen, A., 2017. Automatically generating release notes: An industrial case study. In: 2017 IEEE/ACM 39th International Conference on Software Engineering. (ICSE), pp. 461–472.

Khalajzadeh, H., Shahin, M., Obie, H.O., Agrawal, P., Grundy, J., 2022. Supporting developers in addressing human-centric issues in mobile apps. IEEE Trans. Softw. Eng. 49 (4), 2149–2168.

Kumari, M., Sharma, M., Singh, V.B., 2018. Severity assessment of a reported bug by considering its uncertainty and irregular state. Int. J. Open Source Softw. Processes 9 (-), 20–46. http://dx.doi.org/10.4018/IJOSSP.2018100102.

Lamkanfi, A., Demeyer, S., Giger, E., Goethals, B., 2010. Predicting the severity of a reported bug. In: 2010 7th IEEE Working Conference on Mining Software Repositories. (MSR 2010), pp. 1–10. http://dx.doi.org/10.1109/MSR.2010.5463284.

Lamkanfi, A., Demeyer, S., Soetens, Q.D., Verdonck, T., 2011. Comparing mining algorithms for predicting the severity of a reported bug. In: 2011 15th European Conference on Software Maintenance and Reengineering. pp. 249–258. http://dx.doi.org/10.1109/CSMR.2011.31.

Lin, Y., Wang, S., Hui, G., Chen, X., Xing, Z., Chen, S., 2018. Sentiment analysis of commit comments in GitHub: An empirical study. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications. (SEAA), pp. 195–198.

Menzies, T., Marcus, A., 2008. Automated severity assessment of software defect reports. In: 2008 IEEE International Conference on Software Maintenance. pp. 346–355. http://dx.doi.org/10.1109/ICSM.2008.4658083.

Moran, K., 2015. Enhancing android application bug reporting. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. In: ESEC/FSE 2015, ACM, New York, NY, USA, pp. 1045–1047. http://dx.doi.org/10.1145/2786805.2807557, URL http://doi.acm.org/10.1145/2786805.2807557.

Mozilla, 2018. Bugzilla issue tracker. https://www.bugzilla.org/, Nov, 2018.

Oliaee, A.H., Das, S., Liu, J., Rahman, M.A., 2023. Using bidirectional encoder representations from transformers (BERT) to classify traffic crash severity types. Natural Lang. Process. J. 3, 100007.

Ouyang, X., Zhou, P., Li, C.H., Liu, L., 2015. Sentiment analysis using convolutional neural network. In: 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing. pp. 2359–2364. http://dx.doi.org/10.1109/CIT/IUCC/DASC/PICOM.2015.349.

Ramay, W.Y., Umer, Q., Yin, X.C., Zhu, C., Illahi, I., 2019. Deep neural network-based severity prediction of bug reports. IEEE Access 7, 46846–46857. http://dx.doi.org/10.1109/ACCESS.2019.2909746.

Saga, T., Tanaka, H., Iwasaka, H., Nakamura, S., 2022. Multimodal prediction of social responsiveness score with BERT-based text features. IEICE Trans. Inf. Syst. 105 (3), 578–586.

Sharma, M., Bedi, P., Chaturvedi, K.K., Singh, V.B., 2012. Predicting the priority of a reported bug using machine learning techniques and cross project validation. In: 2012 12th International Conference on Intelligent Systems Design and Applications. (ISDA), pp. 539–545. http://dx.doi.org/10.1109/ISDA.2012.6416595.

Sharma, M., Kumari, M., Singh, R.K., Singh, V.B., 2014. Multiattribute based machine learning models for severity prediction in cross project context. In: Computational Science and Its Applications – ICCSA 2014. Springer International Publishing, Cham, pp. 227–241.

Singh, J.P., Irani, S., Rana, N.P., Dwivedi, Y.K., Saumya, S., Roy, P.K., 2017. Predicting the “helpfulness” of online consumer reviews. J. Bus. Res. 70, 346–355.

Tian, Y., Lo, D., Sun, C., 2012. Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In: Proceedings of the 2012 19th Working Conference on Reverse Engineering. WCRE ’12, IEEE Computer Society, Washington, DC, USA, pp. 215–224. http://dx.doi.org/10.1109/WCRE.2012.31, URL http://dx.doi.org/10.1109/WCRE.2012.31.

Tian, Y., Lo, D., Xia, X., Sun, C., 2015. Automated prediction of bug report priority using multi-factor analysis. Empir. Softw. Eng. 20 (5), 1354–1383. http://dx.doi.org/10.1007/s10664-014-9331-y.

Uddin, J., Ghazali, R., Mat Deris, M., Naseem, R., Shah, H., 2016. A survey on bug prioritization. In: Artificial Intelligence Review, Vol. 47.

Umer, Q., Liu, H., Sultan, Y., 2018. Emotion based automated priority prediction for bug reports. IEEE Access 6, 35743–35752. http://dx.doi.org/10.1109/ACCESS.2018.2850910.

Xia, X., Lo, D., Wang, X., Zhou, B., 2013. Accurate developer recommendation for bug resolution. In: 2013 20th Working Conference on Reverse Engineering. (WCRE), pp. 72–81. http://dx.doi.org/10.1109/WCRE.2013.6671282.

Xu, S., Zhang, C., Hong, D., 2022. BERT-based NLP techniques for classification and severity modeling in basic warranty data study. Insurance Math. Econom. 107, 57–67.

Yang, G., Baek, S., Lee, J.-W., Lee, B., 2017. Analyzing emotion words to predict severity of software bugs: A case study of open source projects. In: Proceedings of the Symposium on Applied Computing. SAC ’17, ACM, New York, NY, USA, pp. 1280–1287. http://dx.doi.org/10.1145/3019612.3019788, URL http://doi.acm.org/10.1145/3019612.3019788.

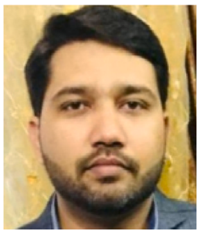
Yu, S., Fang, C., Zhang, Q., Cao, Z., Yun, Y., Cao, Z., Mei, K., Chen, Z., 2023. Mobile app crowdsourced test report consistency detection via deep image-and-text fusion understanding. IEEE Trans. Softw. Eng..

Yu, L., Tsai, W.-T., Zhao, W., Wu, F., 2010. Predicting defect priority based on neural networks. In: Cao, L., Zhong, J., Feng, Y. (Eds.), Advanced Data Mining and Applications. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 356–367.

Zhang, T., Yang, G., Lee, B., Chan, A.T.S., 2015. Predicting severity of bug report by mining bug repository with concept profile. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing. SAC ’15, ACM, New York, NY, USA, pp. 1553–1558. http://dx.doi.org/10.1145/2695664.2695872, URL http://doi.acm.org/10.1145/2695664.2695872.

**Asif Ali** received the B.S. degree in computer science from COMSATS University Islamabad, Pakistan, in 2015, the M.S. degree in computer science from the information Technology University, Lahore Punjab, Pakistan, in 2018, respectively, and the Ph.D. degree from Beijing Institute of Technology, China. He is currently working as lecturer with the Computer Science Department, COMSATS University Islamabad, Vehari Campus, Pakistan. His research interests include machine learning, data mining, software maintenance, and developing practical tools to assist software engineers for developing mobile Applications.

**Yuanqing Xia** (Senior Member, IEEE) received the Ph.D. degree in control theory and control engineering from Beihang University, Beijing, China, in 2001. He was a Research Fellow in several academic institutions, from 2002 to 2008, including the National University of Singapore and the University of Glamorgan, U.K. Since 2004, he has been with the Beijing Institute of Technology (BIT), China, where he is currently a Full Professor and the Dean of the School of Automation. He is also the Director of the Specialized Committee on Cloud Control and Decision, Chinese Institute of Command and Control (CICC). He has published 16 monographs in Springer, John Wiley, and CRC, and more than 500 papers in international scientific journals. He has been a highly cited scholar, since 2014, by Elsevier. His research interests include cloud control systems, networked control systems, robust control, signal processing, active disturbance rejection control, and flight control. He is a member of the 8th Disciplinary Review Group, Academic Degrees Committee, State Council, and the Big Data Expert Committee of the Chinese Computer Society. He was granted by the National Outstanding Youth Foundation of China, in 2012, and was honored as the Yangtze River Scholar Distinguished Professor, in 2016, and the leading talent of the Chinese ten thousand talents program. He received the Second Award of the Beijing Municipal Science and Technology (No. 1), in 2010 and 2015, the Second National Award for Science and Technology (No. 2), in 2011, the Second Natural Science Award of the Ministry of Education (No. 1), in 2012 and 2017, and the Second Wu Wenjun Artificial Intelligence Award, in 2018 (No. 1). More than five of his students have obtained excellent doctoral thesis awards from the Chinese Association of Automation or the Chinese Institute of Command and Control. He is the Vice-Chairperson of the Internet of Things Working Committee, Chinese Institute of Instrumentation. He is the Deputy Editor of the Journal of Beijing Institute of Technology. He is an Associate Editor of Acta Automatica Sinica, International Journal of Automation and Computing, Guidance, Navigation, and Control: Theory and Applications.



**Qasim Umer** is working as a postdoc researcher at the Department of Computer Science, Hanyang University, Seoul, South Korea, and affiliated as an Assistant Professor with the Department of Computer Sciences, COMSATS University Islamabad, Vehari Campus, Pakistan. He received B.S. degree in Computer Science from Punjab University, Pakistan in 2006, M.S. degree in .Net Distributed System Development from University of Hull, UK in 2009, M.S. degree in Computer Science from University of Hull, UK in 2013, and Ph.D. degree from Beijing Institute of Technology, China. He is particularly interested in machine/deep learning, NLP, and IoTs. He is also interested in developing practical tools to assist software engineers.

**Mohamed Osman** received his M.S. degree in control from the University of Karari in 2019 and is doing Ph.D. degree in control science and engineering from the Beijing Institute of Technology, China since 2019. His research interests include system identification, data driven control, model predictive control, and UAVs.