

SENTIMENT-BASED MACHINE LEARNING AND LEXICON-BASED APPROACHES FOR PREDICTING THE SEVERITY OF BUG REPORTS

ALADDIN BAARAH¹, AHMAD ALOQAILY², ZAHER SALAH³, ESRA'A ALSHDAIFAT⁴

¹Department of Software Engineering, Faculty of Prince Al-Hussein Bin Abdallah II For Information Technology, The Hashemite University, P.O. Box 330127, Zarqa 13133, Jordan

²Department of Computer Science and its Applications, Faculty of Prince Al-Hussein Bin Abdallah II For Information Technology, The Hashemite University, P.O. Box 330127, Zarqa 13133, Jordan

^{3,4}Department of Computer Information systems, Faculty of Prince Al-Hussein Bin Abdallah II For Information Technology, The Hashemite University, P.O. Box 330127, Zarqa 13133, Jordan

Email: ¹aladdin.baarah@hu.edu.jo, ²aloqaily@hu.edu.jo, ³zaher@hu.edu.jo, ⁴esraa@hu.edu.jo

ABSTRACT

Fixing bug reports is an important activity performed during software maintenance. End-users and software developers report bugs related to open and closed-source projects through a bug tracking system. They should describe the bug reports carefully, mainly when they assign the severity of the bug. Thus, assigning incorrect severity levels will postpone the fixing order of critical bugs. Many works have been proposed using various machine learning algorithms to classify the severity of bug reports. However, few research works have considered the analysis of reporters sentiments expressed in the summary description of bug reports to predict the bug severity. In this paper, the analysis of the reporters sentiments has been considered and incorporated into the severity prediction process. More specifically, sentiment-based approaches have been proposed, namely machine learning and lexicon-based approaches for predicting the severity of bug reports. SentiWordNet dictionary is used to identify the bug reports sentiment terms and compute their associated sentiment scores. The proposed sentiment-based approaches have been applied and evaluated on a bug reports dataset related to closed-source projects extracted from the JIRA bug tracking system. The results of sentiment-based machine learning and lexicon-based approaches are compared and reported. The results have shown that the Logistic Model Trees (LMT) model outperforms other sentiment-based models, including the lexicon-based model. The reported experimental results also revealed that the lexicon-based approach is not effective for bug severity prediction. However, the sentiment-based machine learning approach significantly improves the severity prediction of bug reports compared to the lexicon-based approach (baseline approach). The severity prediction accuracy has been remarkably enhanced from 53% for lexicon-based to 87.14%. Likewise, the F-Measure of the severity prediction has been improved from 0.65 for lexicon-based to 0.91 after applying the machine learning approach.

Keywords: *Software Maintenance; Bug Report; Severity Prediction; Sentiment Analysis; Machine Learning Algorithms; Lexicon-Based; Sentiment-Based Approach.*

1 INTRODUCTION

A software bug report is an essential software artifact used to execute tasks (e.g., bug severity classification and bug fixing) in the software maintenance phase during the development of open and closed-source projects. End-users and developers report large amounts of bug reports related to medium or large-scale software systems

through bug tracking systems (BTS). Thus, fixing bug reports has become a significant task, especially when the severity level of bug reports is high [1, 2]. In software maintenance, developers aim to enhance the next software release by fixing bugs collected from BTS, such as Bugzilla [3] and JIRA [4]. Therefore, they permit software users to participate and collaborate in software improvement by

reporting and submitting bugs using a particular BTS [5].

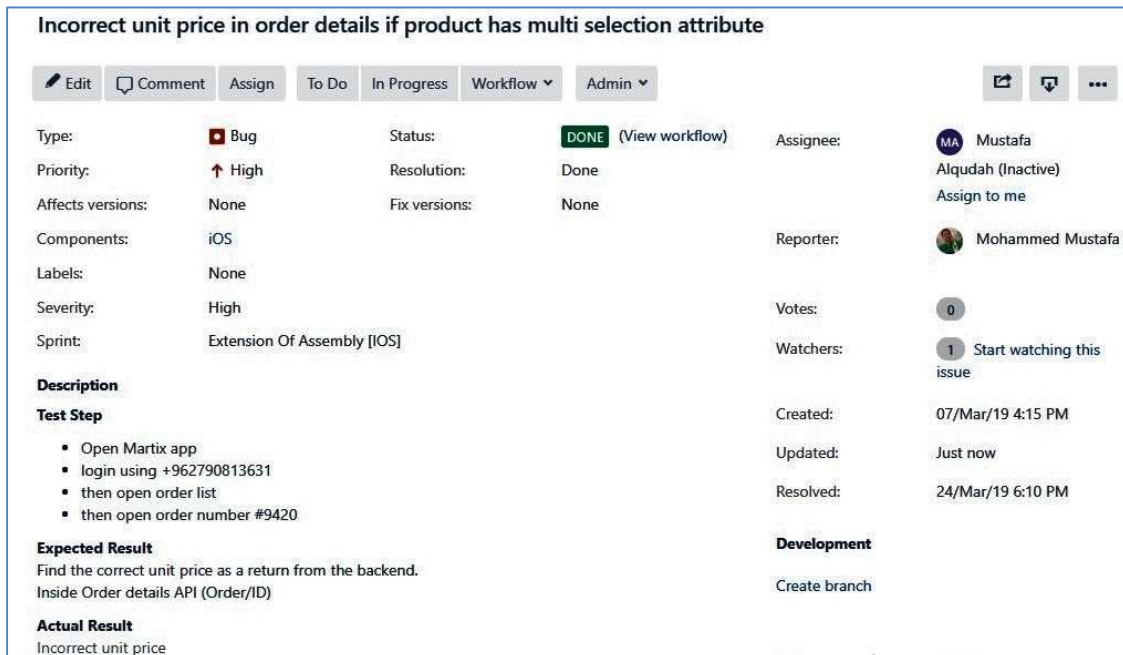
Once a bug has been detected in the software, the reporters use a specific bug report form, according to the used BTS, to fill the contents of the bug report based on their experience and background knowledge. A bug report includes unstructured text written in natural language to represent the summary and detailed description fields. Also, it contains other significant fields, such as severity and priority. The severity level determines how quickly the bug should be fixed. For instance, if the bug report is classified as severe, it should have immediate attention and should be resolved as soon as possible because it impacts the performance and functionality of the system. On the other hand, the priority level identifies the importance and fixing order of the bug report [6].

The bug severity levels and their definitions depend on the utilized BTS system [7]. In Bugzilla, the severity points out the effect of the bug on the system functionalities. It is expressed by six different levels, from Blocker (high severity) to trivial (low severity). While in JIRA, the severity is referred to as a priority and varies from Blocker to minor. It denotes the significance of the bug relative to other bug reports. In other words, the bug that is assigned a Blocker level (highest priority) requires urgent attention and has to be resolved immediately.

A software development company that adopts JIRA can customize bug reports related to closed-source projects by adding new fields or updating existing fields. The INTIX software development

Company, which the proposed methodology has been applied to their closed-source dataset, has made a slight change in the bug report format. Even though the priority field in JIRA represents severity, the software development team has explicitly added a new field called severity and changed the priority definition. The development team has also added five new levels for the severity field (highest, high, medium, low and lowest). Figure 1 shows an example of the customized JIRA bug report form adopted by INTIX Company.

When bug reports are submitted to BTS, they are initially inspected and assessed by a group of bug triagers or development team. In this stage, the triager reviews the reported bug fields and confirms that the severity level remains unchanged. Even though there are general rules on determining the proper severity level of a bug report, in many cases, the users may either assign an incorrect severity level for a given bug report or leave it blank. This is due to their lack of experience and limited domain knowledge [8]. Since there are many bug reports submitted daily through BTS, the triagers spend a considerable amount of time examining these bug reports manually to verify the severity level and may reassign a suitable severity level if required. As a result, the triagers or development team effort will increase, causing delays in processing and fixing critical bug reports [9, 10]. Therefore, to alleviate the burden on bug triagers and reduce errors regarding severity level assignment, the bug report severity prediction should be automated.



Incorrect unit price in order details if product has multi selection attribute

Edit Comment Assign To Do In Progress Workflow Admin

Type: Bug Status: DONE (View workflow) Assignee: Mustafa
 Priority: High Resolution: Done Assign to me
 Affects versions: None Fix versions: None
 Components: iOS Reporter: Mohammed Mustafa
 Labels: None Votes: 0
 Severity: High Watchers: 1 Start watching this issue
 Sprint: Extension Of Assembly [IOS] Created: 07/Mar/19 4:15 PM
 Updated: Just now
 Resolved: 24/Mar/19 6:10 PM

Description

Test Step

- Open Martix app
- login using +962790813631
- then open order list
- then open order number #9420

Expected Result
Find the correct unit price as a return from the backend. Inside Order details API (Order/ID)

Actual Result
Incorrect unit price

Development
Create branch

Figure 1: An Example of a Customized JIRA Bug Report Form

Several research works have been proposed to tackle the problems mentioned above [6, 11-16]. Most of the proposed studies have used well-known machine learning algorithms such as Support Vector Machine, Decision Trees, Naïve Bayes, K-Nearest Neighbor. However, none of these studies consider the reporters sentiments in predicting the severity of bug reports. According to the study conducted by Umer, et al. [5], the number of negative sentiment words written by the reporters in the severe bug reports is higher than in the non-severe bug reports. In other words, the severity of bug reports assigned by the bug reporters depends on their emotional expressions written in the summary and description fields of the reported bugs. Negative sentiment words (e.g., crash, error, wrong and incorrect) expressed by the bug reporter may indicate that the bug is severe and requires urgent action. Therefore, emotional expressions could be significant in predicting the severity levels of bug reports.

This paper proposes a sentiment-based methodology that takes the reporters sentiments expressed in the summary field of bug reports into account. Two sentiment-based approaches are employed, namely: sentiment-based machine learning and lexicon-based approaches. Concerning sentiment-based machine learning, five well-known machine learning algorithms are utilized, which are Naïve Bayes (NB), Logistic Regression, Vote-Based, Support Vector Machine (SVM), Random Forest (RF) and Logistic Model Tree (LMT). The sentiment-based approaches depend on the popular SentiWordNet lexicon to identify the sentiment terms and calculate their associated sentiment scores. In order to evaluate the performance of the proposed methodology, the sentiment-based approaches are applied on a dataset related to closed-source projects developed by a Jordanian software development Company called INTIX. This dataset is extracted from the JIRA repository.

In summary, this study makes the following contributions:

1. Sentiment-based machine learning and lexicon-based approaches are proposed to predict the severity levels of bug reports.
2. The emotions of bug reporters expressed in the summary field of the bug reports are considered and incorporated in the bug severity prediction process.
3. The sentiment analysis process in the proposed sentiment-based approaches is different from other similar studies mentioned in the literature.

This paper is organized as follows. Section 2 describes the works related to the proposed sentiment-based methodology. Section 3 demonstrates, in detail, the proposed methodology. Then, Section 4 discusses the experimental results.

In Section 5, the potential threats to the validity of the proposed methodology are introduced. Finally, Section 6 concludes the paper and suggests future works.

2 RELATED WORKS

Machine learning and natural language processing (NLP) techniques have recently been employed in software engineering to automate many software maintenance tasks, such as automating bug reports severity prediction. Numerous research works have been conducted to predict the severity level of reported software bugs. The majority of the works have employed well-known machine learning algorithms such as SVM, Decision Trees, Naïve Bayes, Naïve Bayes Multinomial, and K-Nearest Neighbor (K-NN). However, a small number of studies have considered the sentiments of bug reporters to predict the severity of bug reports.

One of the first attempts to predict the severity of the software bugs was suggested by Menzies and Marcus [17]. They developed a new technique called SEVERIS to help test engineers set the appropriate severity level of a particular bug while validating NASA's closed source projects. SEVERIS mainly relied on both text mining techniques to process the text description of the defected bugs and the rule learning approach to classify the bug reports using the RIPPER rule learner method.

Later on, Lamkanfi, et al. [18] and Lamkanfi, et al. [15] conducted two studies based on the work of Menzies and Marcus [17] to predict the severity of the newly submitted bug. They applied machine learning algorithms on the short textual description (i.e., text summary) of the historical bug reports stored in the Bugzilla bug tracking system related to open source projects. In both studies, the authors deduced that the short text of the historical bug reports could be utilized to precisely predict the coarse-grained severity level (i.e., severe and non-severe) of the submitted bugs using machine learning algorithms. While in a follow-up study of Lamkanfi, et al. [15], they applied four machine learning algorithms, particularly Naïve Bayes, Naïve Bayes Multinomial, K-NN and SVM, on different datasets related to two open-source projects: Eclipse and GNOME. According to the experimental results of their study, Naïve Bayes Multinomial showed better performance compared to other algorithms.

A work proposed by Tian, et al. [13] used different attributes of historical bug reports (e.g., summary text, description text and product) to predict the fine-grained severity level (i.e., Critical, Major, Minor and Trivial). They proposed an algorithm based on a combination of the BM25F_{ext} similarity method to compute the similarity between two bug reports and K-NN, taking into account duplicate bugs. Various datasets related to Mozilla,

OpenOffice and Eclipse open-source projects were utilized to validate their approach.

An approach of comparing ten machine learning algorithms, namely RF, SVM, Naïve Bayes, K-NN, Decision Tree, Boosting, Bagging, Stabilized Linear Discriminant Analysis (SDLA), Generalized Linear Model (Glmnet), Maximum Entropy (MAXENT) was conducted by Kaur and Jindal [19] on thirteen datasets of Apache projects extracted from the JIRA repository. They claimed that none of the following four algorithms, SDLA, Glmnet, MAXENT and Bagging, were used together in a single work. Based on their results, the Boosting algorithm performed the best in terms of accuracy, whereas SLDA and Glmnet were given the least results.

A recent study conducted by Tan, et al. [1] differs from other studies in the literature. They used the question and answer posts from Stack Overflow related to open-source projects Eclipse, Mozilla, and GCC and integrated them with the bug reports of these projects. The reason for adding this additional detailed information was to make the bug reports dataset better. For fine-grained bug severity prediction, the authors employed the Logistic Regression machine learning algorithm. The results showed that their approach outperformed other machine learning algorithms, including Naïve Bayes, KNN and Long Short-Term Memory (LSTM) when an enhanced version of the bug reports dataset was taken into consideration.

Several studies adopted topic modeling and similarity functions to automate the severity prediction [8, 10, 20]. All these studies employed Latent Dirichlet Allocation (LDA) to extract the topics built from the summary and description fields of bug reports available in a given dataset. So, the bug reports which belong to the same topic have similar textual contents. In terms of computing the degree of similarity between bug reports, Yang, et al. [10] and Yang, et al. [20] used the KL-divergence similarity measure, while Zhang, et al. [8] proposed a new similarity algorithm called REP_{topic} .

Few researchers proposed to include the analysis of reporters sentiments to predict the severity of reported bugs [21-23]. These sentiments are expressed and presented clearly in the summary description of the bug reports.

One of the first studies in this domain was conducted by Yang, et al. [23]. They investigated the impact of emotional words on predicting the severity of bug reports from three open-source projects. In summary, the authors proposed a modified version of the original Naïve Bayes Multinomial algorithm and called it EWD-Multinomial. Their methodology was based on analyzing the sentiment words according to the well-known sentiment lexicon called SentiWordNet. According to their experimental results, their approach performed

better than the original Naïve Bayes Multinomial and the work proposed by Lamkanfi, et al. [18].

Later, Yang, et al. [22] conducted another study using a different sentiment-based approach. In their work, the authors reported a new concept by finding out the emotional similarity between the reported bug and the historical bug reports related to different open source projects using the Emotion-based Smoothed Unigram Model and KL-divergence. Like the work of Yang, et al. [23], the SentiWordNet lexicon was employed in the emotion similarity analysis phase. The authors also modified the original Naïve Bayes Multinomial and proposed a new Emotion Similarity (ES) Multinomial algorithm to predict the severity level of bug reports. In general, their research findings indicated that their methodology had better results compared to other works proposed by Lamkanfi, et al. [18], Yang, et al. [23] and Yang, et al. [22].

Ramay, et al. [21] carried out comparative research. However, besides the sentiment analysis of reporters emotions expressed in the bug reports summary field, the authors were the first to employ a deep learning algorithm to predict the bug reports severity levels. Unlike the work proposed by Yang, et al. [23] and Yang, et al. [22], where SentiWordNet was used for emotion analysis, the authors exploited a different popular sentiment dictionary called Senti4SD to compute the sentiment score for each bug report of a particular dataset extracted from Bugzilla. Their work revealed a considerable improvement compared to EWD-Multinomial proposed by Yang, et al. [23].

Umer, et al. [5] conducted another study that considered emotion analysis using the SentiWordNet dictionary. They applied emotion analysis on bug reports related to the Eclipse open-source project to predict the priority of the bug reports. In their approach, the emotion-score of bug reports and the extracted features were used to train the SVM algorithm. The authors performed several experiments, and they concluded that their work performed better than other approaches listed in the literature. Their approach also outperformed other machine learning algorithms, including Naïve Bayes, Multinomial Naïve Bayes and Linear Regression.

This study is similar to the studies mentioned above, where the emotion analysis is exploited for severity prediction. However, this study is different from other studies [5, 21-23]. First, this study is applied to a dataset related to closed-source projects developed by a private company. Second, sentiment analysis is performed differently to train the machine learning, where the sentiment score for each sentiment term for all bug reports is submitted to the machine learning algorithms, instead of submitting the sentiment score of each bug report as followed

by other studies. Finally, this study has employed the lexicon-based as a baseline approach to compare the proposed sentiment-based machine learning approach.

3. SENTIMENT-BASED SEVERITY PREDICTION METHODOLOGY

We propose an approach based on sentiment analysis and machine learning algorithms to predict the severity levels of bug reports. The proposed approach is a binary classification technique in which a new bug report is classified into two severity levels (i.e., severe or non-severe). The proposed sentiment-based severity prediction methodology is shown in Figure 2. The procedure of predicting the severity of a newly reported bug is briefly illustrated

as follows: first, the historical bug reports from the JIRA bug tracking system are extracted and the dataset is prepared. Second, NLP techniques to pre-process the short description text (i.e., summary) of each bug report in the dataset are applied. Third, a feature matrix after performing the pre-processing stage is constructed. Fourth, the sentiment terms for each bug report according to the SentiWordNet lexicon are identified. Then the sentiment score for each sentiment term presented in the feature matrix is computed. Finally, the sentiment-based classifiers to predict the severity level of the newly reported bugs are trained and tested. In this phase, the input to the sentiment-based classifier is a collection of vectors representing bug reports sentiment terms with their sentiment scores.

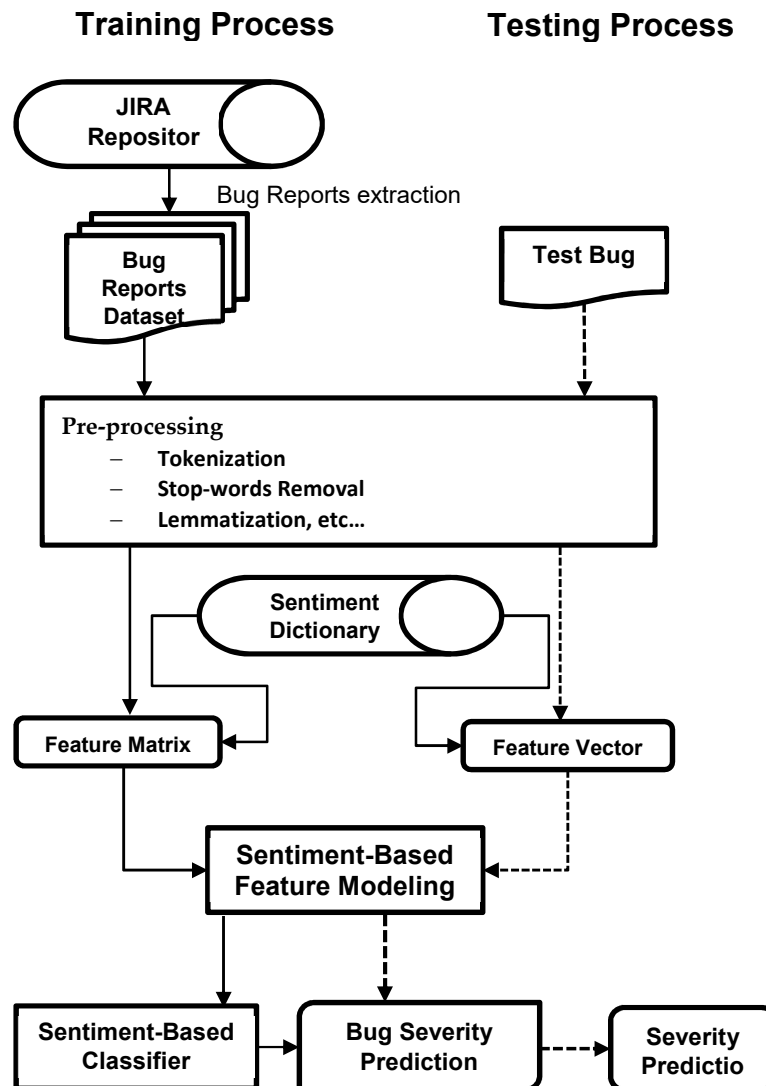


Figure 2: Sentiment-Based Severity Prediction Methodology

3.1 Data Acquisition

In general, software bugs are reported during software maintenance and managed and tracked using a particular bug tracking system. In this study, the dataset is prepared and constructed by extracting bug reports from the JIRA bug tracking repository. The software bug reports are related to different closed-source projects developed by a Jordanian Company called INTIX located in Amman, Jordan [11].

When software developers report a particular bug, they choose one of five severity levels to determine how severe the bug is. The severity levels are categorized into five classes: highest (i.e., the most severe), high, medium (i.e., normal), low and lowest (i.e., the least severe). In this work, low and lowest severity levels are considered as non-severe bugs, whereas high and highest severity levels are labeled as severe bugs. However, we suggest excluding bug reports whose severity levels are classified as normal. As investigated with the development team, the normal severity level seems to be the default option for the bug reporters, especially when it is challenging to decide the appropriate severity level for each bug report. Therefore, the bug reports in the dataset categorized as normal are discarded from the final utilized dataset.

A software bug report, *sbr*, is composed of two main fields; a short description of the bug, *sd*, represented by a summary text and a severity level, *sl*, associated with each bug. In short, a software bug report can be formalized as follows:

$$sbr = \langle sd, sl \rangle \quad (1)$$

where $sl \in \{severe, non_severe\}$

Table 1 provides a statistical summary of the utilized closed-source dataset. The bug reports created between May 2016 and March 2018 are only included.

Table 1: Statistical Summary of Closed-Source Dataset

Statistical description	Value
Number of all bug reports	1164
Number of severe bugs	851
Number of non-severe bugs	313
Number of all terms	9016
Number of unique terms	926

After ignoring the bug reports classified as normal, the total number of bug reports used in the experiments is 1164. About 27% of the bug reports are classified as non-severe bugs and the remaining

bug reports are classified as severe. The number of all terms found in the short description of all bug reports is 9016. After applying NLP techniques, the number of distinct terms extracted from the bug reports short description is decreased remarkably. It is found that there are 926 distinct terms.

3.2 Pre-processing

In this step, NLP techniques are applied to the unstructured text (i.e., summary description field) of the bug reports dataset. In this work, NLP techniques are employed to process the textual part of the dataset. The goal of NLP is to pre-process and transform the textual representation of bug reports into a collection of distinct terms or words, which is called bag-of-words (BOW). The text pre-processing phase incorporates tokenization, stop-word removal and lemmatization approaches.

The first step in pre-processing is tokenization, in which the summary description of each bug report is split into a set of terms or words. The second step in pre-processing is stop-word removal, where the unnecessary words, which do not play a part in predicting the severity of reported bugs, are discarded. In other words, constructive words such as “a”, “an”, “the”, “that”, “on”, “in” and many others are commonly used in describing the bug reports and do not convey specific information. Therefore, stop-words and special characters from the bug reports dataset are removed as they reduce the precision of anticipating the severity of newly reported bugs. This process is performed based on a predefined list of stop-words implemented in Weka called *Rainbow*.

The final step of the pre-processing phase is lemmatization. Usually, the words in the summary field of bug reports written by bug reporters can be expressed differently and appear in several styles. However, they still carry the same meaning. After applying the two previous steps, the remaining words are reduced and transformed into their root. As a result, the duplication of words that have a similar root will be avoided. For example, the words “failure”, “failures”, “fails”, “failed” and “failings” are converted to the root word “fail”. In this work, the Porter stemming algorithm is utilized [24]. It is the most commonly known algorithm that has been used in the NLP domain to reduce each word to its basic form.

The following definition depicts the pre-processing of a bug report *sbr*:

$$sbr' = \langle sd', sl \rangle \quad (2)$$

$$sd' = \langle t_1, t_2, \dots, t_n \rangle$$

where, *sbr'* is the pre-processed software bug report, *sd'* is the pre-processed summary description of a software bug report *sbr*, *sl* represents the

severity level associated with the bug report sbr and t_1, t_2, \dots, t_n represent the terms or words resulted from applying the three steps of pre-processing of the software bug report sbr .

After applying the pre-processing phase, each bug report is represented as a single vector. Assume we have r bug reports, in this case, we will have r vectors of length n , where n represent all the unique words occurring in all bug reports. In other words, at the end of the pre-processing stage, we will have a feature matrix of size $r \times n$.

3.3 Sentiment-Based Feature Modeling

As shown in the proposed methodology, the next phase is to perform sentiment analysis as the bug reports could be expressed in a sentiment manner (i.e., using sentiment terms) by the bug reporters. The process of sentiment analysis entirely relies on the sentiment dictionary. In this work, the SentiWordNet is employed. It is a popular and publicly available lexical tool used widely in the sentiment analysis context [25].

SentiWordNet comprises of sentiment words and their associated positive score $PosScore$ and negative score $NegScore$. The net sentiment score $NetSentiScore$ for each sentiment word in the corpus is calculated by computing the difference between $PosScore$ and $NegScore$. In other words, if the value of $PosScore$ is higher than $NegScore$, then $NetSentiScore$ is positive. Otherwise, if the value of $NegScore$ is higher than $PosScore$, then $NetSentiScore$ is negative.

In this phase, each pre-processed bug report is exploited to find out whether each word is sentiment or not. To determine a set of sentiment terms $SentiTermSet$ for each sbr' , we compare the pre-processed bug report words sbr' and sentiment words available in the sentiment dictionary. Once they match, the sentiment term $SentiTerm$ is added to $SentiTermSet$ as depicted in the following equation:

$$SentiTermSet = \left\{ SentiTerm \left| \begin{array}{l} term \in sentiList \\ where \ term = \ words_{sentimentwords} \end{array} \right. \right\} \quad (3)$$

where, $SentiTermSet$ is a collection of terms that belong to $sentiList$, the $term$ represents the word in sbr' , $SentiTerm$ denotes sentiment terms in the pre-processed bug report sbr' and $sentiList$ indicates the sentiment words in the sentiment dictionary.

After identifying a set of sentiment terms, the pre-processed software bug report, sbr' , represented as follows:

$$sbr'' = \langle SentiTermSet, sl \rangle \quad (4)$$

$$SentiTermSet = \langle SentiTerm_1, \dots, SentiTerm_n \rangle$$

After that, the SentiWordNet lexicon is used to compute the sentiment score $SentiTermScore$ for

each $SentiTerm$ included in $SentiTermSet$ by calculating the $NetSentiScore$ as follows:

$$SentiTermScore(SentiTerm_i) = \begin{cases} -NetSentiScore_i, & PosScore_i < NegScore_i \\ NetSentiScore_i, & PosScore_i > NegScore_i \end{cases} \quad (5)$$

$$NetSentiScore_i = \begin{cases} PosScore_i - NegScore_i, & PosScore_i > NegScore_i \\ NegScore_i - PosScore_i, & NegScore_i > PosScore_i \end{cases}$$

The sentiment score $SentiTermScore$ has two possible values; positive and negative. When $SentiTermScore$ is positive, this indicates that $SentiTerm$ has a positive sentiment. Otherwise, it has a negative sentiment.

3.4 Sentiment-Based Approaches

The methodology uses two different sentiment-based approaches to predict the severity level of bug reports: a lexicon-based approach (baseline approach) and a sentiment-based classifier using different machine learning algorithms.

3.4.1 Lexicon-based approach

The lexicon-based approach aims to analyze the sentiment of a bug report to predict its severity level (i.e., severe or non-severe). In particular, the overall sentiment score of a given bug report articulates its severity. This study assumes that a bug report is classified into a severe level when the total sentiment score of the bug is negative. It is clear that negative sentiment terms expressed by the reporters signify that the bugs require immediate attention and should be resolved instantly. On the other hand, a bug report is assigned a non-severe level when the total sentiment score of the bug report is positive. Because the bug reporter positively describes and summarizes the reported bug using positive sentiment terms, in this case, the bug report can be postponed and fixed later.

Given a pre-processed software bug report represented as a set of sentiment terms, we target to calculate the sentiment score for each bug report $SentiBugScore$ and classify it into positive or negative sentiment. Ultimately, a bug report with negative sentiment is assigned a severe level, whereas a bug report with positive sentiment is given a non-severe level. The sentiment score of the bug report is computed by finding the total sum of sentiment scores of all sentiment terms present in the pre-processed bug report sbr'' , as described in the equation below.

$$SentiBugScore(sbr) = \sum_{i=1}^n SentiTermScore(SentiTerm_i) \quad (6)$$

Next, the severity level of the given bug report is assigned according to its sentiment score. There are three possible values regarding the sentiment score of a bug report ($SentiBugScore$). It can be negative, positive or zero. When the sentiment score of the bug report is negative, it means that the given bug report

belongs to a severe class. In contrast, when the *SentiBugScore* is positive, this bug report is categorized as non-severe. These two cases are depicted in the equation below.

$$Severity(sbr_i) = \begin{cases} severe & , SentiBugScore_i < 0 \\ non_severe & , SentiBugScore_i > 0 \end{cases} \quad (7)$$

The last case is when the sentiment score of a bug report equals zero. Here, the total number of positive and negative sentiment terms in the pre-processed bug report "*sbr*" must be counted. After that, the severity of the bug report is assigned according to the comparison between the total number of positive terms *CountTermPos* and the total number of negative terms *CountTermNeg*. This comparison is shown in the equation below.

$$Severity(sbr_i) = \begin{cases} severe, & CountTermNeg > CountTermPos \\ non_severe, & CountTermPos > CountTermNeg \end{cases} \quad (8)$$

3.4.2 Sentiment-based Machine Learning Approach

The proposed methodology further utilized a different sentiment-based approach based on the machine learning-algorithms called: sentiment-based classifiers. The work presented here is different from other related works in the literature [5, 21-23], where the authors proposed to pass the total sentiment score of each bug report to different classifiers. While in this proposed approach, the sentiment score of each sentiment term present in the pre-processed bug reports is submitted to several machine learning algorithms.

Initially, to build a sentiment-based prediction model, the bug reports dataset has to be analyzed, then a set of effective sentiment terms for classifying the severity level of the bug report has to be identified. These terms are extracted from each pre-processed software bug report *sbr'*. After that, each sentiment score of the sentiment terms included in the pre-processed bug report is calculated, as mentioned earlier. The potential sentiment scores are zero, positive or negative value. In order to calculate the sentiment score of each sentiment term, first, we have to examine whether the terms are listed in the *SentiWordNet* or not. In case the term does not exist in the lexicon, then that term is not considered a sentiment, and hence a zero value is assigned to it. In contrast, if the term is found, a score is given and the sentiment score is computed as mentioned in equations (4) and (5).

Once the sentiment score of each sentiment term present in bug reports is computed, a sentiment-based feature matrix is constructed to train the sentiment-based classifier. It is composed of *n* columns and *r* rows where the unique features across all *r* pre-processed bug reports are the

columns of the feature matrix, and each software bug report is a row of the feature matrix. Thus, when the pre-processed bug reports have *n* distinct features, we would have an *n*-dimensional feature vector representing each bug report in the dataset. The definition of a feature vector is formalized as follow:

$$sbr = \langle f_1, f_2, \dots, f_n \rangle \quad (9)$$

where *sbr* is a software bug report, *n* represents the total number of distinct features, *f*₁, *f*₂, ... *f*_{*n*} depicts the unique features extracted from all pre-processed bug reports.

Next, in order to fill the feature matrix, each feature vector has to be filled. Therefore, a rule has been defined to fill the values of all features that exist in each feature vector. The feature is assigned a zero value if it does not exist in the corresponding pre-processed bug report. Otherwise, if the feature does belong to the corresponding pre-processed bug report, in this case, the feature may be assigned zero value when the feature is not a sentiment term or assigned a sentiment score (i.e., positive or negative score). As calculated in equations (4) and (5), the sentiment score is used to represent each feature considered as a sentiment term. The equation below expresses how to fill the feature vector.

$$f_i(sbr) = \begin{cases} 0, & \text{if } f_i \notin sbr' \text{ OR } (f_i \in sbr' \text{ AND } f_i \notin SentiTermSet) \\ SentiTermScore(f_i), & \text{if } f_i \in sbr' \text{ AND } f_i \in SentiTermSet \end{cases} \quad (10)$$

In the sentiment-based classifier, different machine learning algorithms are applied, and the goal is to explore which one is best fitted for predicting the correct severity level of bug reports. This study employs the following machine learning algorithms: Naïve Bayes (NB) [26], Random Forest (RF) [27], Logistic Model Trees (LMT) [28], Support Vector Machine (SVM) [5], Logistic Regression and Voting algorithm. The employed Voting algorithm incorporates both SVM and Logistic Regression algorithms to build the sentiment-based model. Both RF and LMT are based on the Decision Tree algorithm. The reason for adopting the algorithms mentioned above is their importance in dealing with unstructured text and their competitive performance, as reported in the literature [29, 30]. All the machine learning algorithms are executed and evaluated using the open-source WEKA software [31].

3.5 Experimental Procedure and Performance Metrics

The proposed sentiment-based approaches are evaluated as follows. First, a bug reports dataset is built. It is extracted from the JIRA repository related to closed-source projects developed by a Jordanian Company called INTIX. Then each bug report in the

dataset is pre-processed, as mentioned in the pre-processing section. After that, the Information Gain (IG) feature selection method is employed to select the top 'N' sentiment features. Finally, the k -fold cross-validation (k -fold CV) approach is applied to the entire dataset to evaluate the performance of sentiment-based machine learning models. This k -fold cross-validation approach is mainly employed to avoid the over-fitting problem.

In this study, 10-fold cross-validation is used to assess the performance of the proposed approach. The original dataset is divided into ten equal folds or sets. In the first iteration ($k=1$), fold number one is used as a testing dataset to validate the performance of the machine learning. The nine remaining folds are used as a training dataset to train the machine learning model. After finishing the first iteration, the performance metrics are measured and retained. In the second repetition ($k=2$), fold number two is used as a test set and the remaining are used as a training set, and the performance metrics are also computed and recorded for the second model. This procedure is iterated until iteration $k = 10$, where each fold of the 10-folds is used as a test dataset. As a result, ten models for a particular machine-learning algorithm are generated. Then, all recorded performance metrics for all ten models are averaged.

The performance of the proposed sentiment-based approaches is measured using well-known performance metrics found in the literature [32]. The four performance measures, Accuracy, Precision, Recall and F-Measure, are commonly used to measure and evaluate the performance of the sentiment-based approaches. The following equations are calculated and reported for the sentiment-based machine learning and lexicon-based prediction models:

$$Accuracy = \frac{BugSeverityTP + BugSeverityTN}{BugSeverityP + BugSeverityN} \quad (11)$$

$$Precision = \frac{BugSeverityTP}{BugSeverityTP + BugSeverityFP} \quad (12)$$

$$Recall = \frac{BugSeverityTP}{BugSeverityTP + BugSeverityFN} \quad (13)$$

$$F-Measure = \frac{2 * Precision * Recall}{Precision + Recall} \quad (14)$$

where,

- BugSeverityP is the number of severe bugs.
- BugSeverityN is the number of non-severe bugs.
- BugSeverityTP is the number of severe bug reports that are correctly classified by the model.
- BugSeverityTN is the number of non-severe bug reports that are correctly classified by the model.

- BugSeverityFP is the number of non-severe bug reports that are incorrectly classified by the model.
- BugSeverityFN is the number of severe bug reports that are incorrectly classified by the model.

Also, another performance metric called Receiver Operating Characteristic (ROC) has been considered [32]. According to the True Positive Rate (TPR) and False Positive Rate (FPR), the ROC curve is plotted. The Area Under Curve (AUC) is a metric that measures the capability of the severity prediction model to differentiate between severe and non-severe bugs. The value of AUC ranges from 0 to 1. A prediction model performs better than others in predicting the correct severity level of a bug report when its AUC value is close to 1.

4. EXPERIMENTAL RESULTS AND DISCUSSION

Usually, the words written by a developer or end-user to report a particular bug reflect their emotions. When the reporters encounter a bug, we assume that they describe the bug using proper sentiment terms. Thus, the severity level assigned for a bug report depends on the reporters sentimental expression. From this perspective, a methodology is proposed to perform sentiment analysis on the bug reports to predict their severity levels.

This work aims to compare the performance of two sentiment-based approaches for predicting the severity of bug reports. These approaches are sentiment-based classifier using machine learning algorithms and lexicon-based approach. The results of the sentiment analysis performed in the sentiment-based feature modeling component of the proposed methodology will be submitted to the sentiment-based approaches for training and testing purposes.

Regarding sentiment-based classifiers, machine learning algorithms (SVM, RF, LMT, NB, Logistic Regression, and Vote-Based) are utilized to investigate the algorithm with the best performance results to predict the bugs severity level. The lexicon-based approach employs the SentiWordNet dictionary to classify the severity of bug reports according to the emotions of bug reporters expressed as unstructured text in bug reports summary description. Both sentiment-based approaches are applied to the same private dataset related to closed-source projects. In this study, the lexicon-based is selected as a baseline approach, and the sentiment-based machine learning approach is compared to the lexicon-based approach in terms of performance evaluation results.

After pre-processing the original dataset, the IG feature selection method is applied to rank all the feature vector terms according to the obtained

scores. In this experiment, the top 50, 100, 150, 200 and 300 sentiment terms are selected to generate different corresponding datasets, and each has 50, 100, 150, 200, and 300-dimensional vectors, respectively. The sentiment-based machine learning algorithms are then applied to the originated datasets for analysis and evaluation using performance metrics. All the experiments related to the machine learning algorithms are performed with different parameter settings using the open-source Weka data mining tool.

4.1 Sentiment-Based Machine Learning Approach Results

The average accuracy, F-Measure and AUC of applying different sentiment-based machine learning algorithms on the entire pre-processed dataset are presented in Table 2. As depicted in Table 2, the accuracy ranges between 76.34% and 84.57%. The Logistic Regression algorithm has the lowest accuracy, while the RF algorithm has the highest accuracy. Since the dataset is imbalanced and approximately 73% of bug reports are classified as severe, the accuracy results of all sentiment-based machine learning algorithms are biased toward the severe class. However, F-Measure and AUC results are encouraging because they take into account the non-severe class that roughly forms 27% of the bug reports. As observed from Table 2, the F-Measure values lie in the range of 0.85 – 0.90, and AUC values fall in the range of 0.59 - 0.88. Thus, F-Measure and AUC performance metrics are not biased towards the severe class even though most bug reports in the dataset are categorized as severe. So, it can be concluded from Table 2 that the RF prediction model performs better than other prediction models in predicting the severity levels of bug reports, as can be seen from the F-Measure and AUC results.

Once the original dataset is pre-processed, the IG feature selection method is utilized to reduce the number of features in the dataset and improve the

performance of sentiment-based prediction models. Besides the original pre-processed dataset, five other datasets with different numbers of selected features are generated in this experiment. The goal is to investigate which dataset with top ‘N’ sentiment terms makes the sentiment-based model performs better than others. The performance results of sentiment-based machine learning algorithms applied on each originated dataset with top-50, top-100, top-150, top-200 and top 300 sentiment terms are presented in Figure 3, Figure 4 and Figure 5. Figure 3 shows the accuracy results of all sentiment-based machine learning models. As observed from Figure 3, the performance of sentiment-based models is improved when the IG feature selection method is employed compared to the performance results of sentiment-based models applied to the original entire pre-processed dataset that contains all features. In particular, as it is evident from Figure 3, the Vote-Based model has the highest accuracy (87.14%) and hence outperforms other sentiment-based models when the top-200 sentiment terms are considered for prediction. On the other hand, when the IG feature selection method is not employed, the maximum accuracy reaches 84.57%. In general, from Figure 3, it can be seen that all the sentiment-based models can exceptionally distinguish the bug reports severity level when top-50 until top-200 sentiment features are taken into account compared to the maximum accuracy (84.57%) when the IG method is not applied. This is indicative from the maximum values of accuracy which are 85.6%, 86.6%, 86.44%, 87.14%, 86.21% for sentiment-based machine learning models corresponding to top-50, top-100, top-150, top-200 and top-300 sentiment terms respectively.

In terms of F-Measure and AUC, the evaluation of sentiment-based machine learning models is depicted in Figure 4 and Figure 5, respectively. These performance metrics are computed for top-50, top-100, top-150, top-200 and top-300 sentiment terms for every sentiment-based model.

Table 2: Results of the Sentiment-Based classifier for all Sentiment Terms of the Dataset.

Sentiment-Based Classifier	Accuracy	F-Measure	AUC
Naïve Bayes (NB)	80.06	0.86	0.84
Random Forest (RF)	84.57	0.90	0.88
Logistic Model Tree (LMT)	83.68	0.89	0.85
Vote-Based	76.82	0.85	0.69
Support Vector Machine (SVM)	77.24	0.87	0.59
Logistic Regression	76.34	0.85	0.70

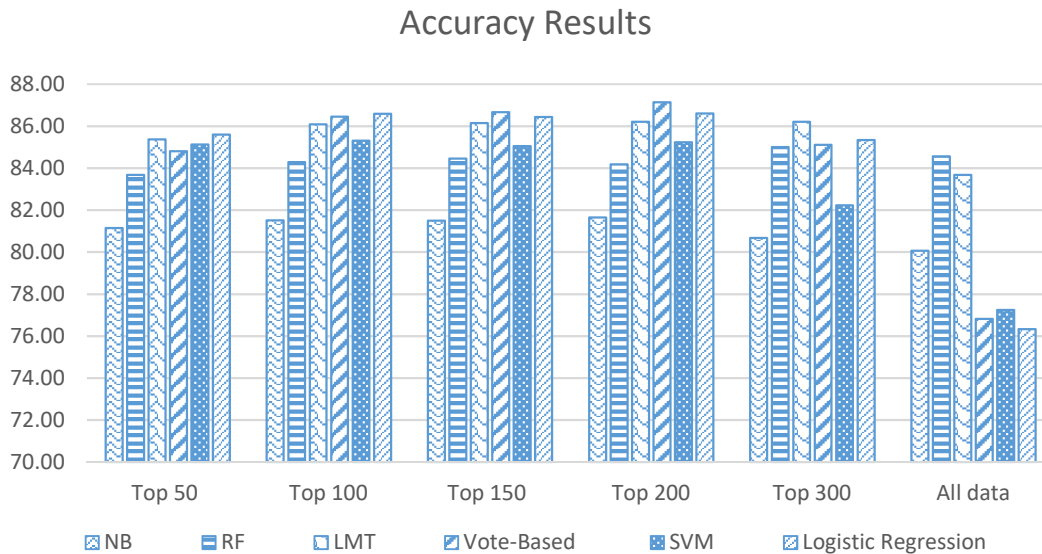


Figure 3: Accuracy of Sentiment-Based Machine Learning Models.

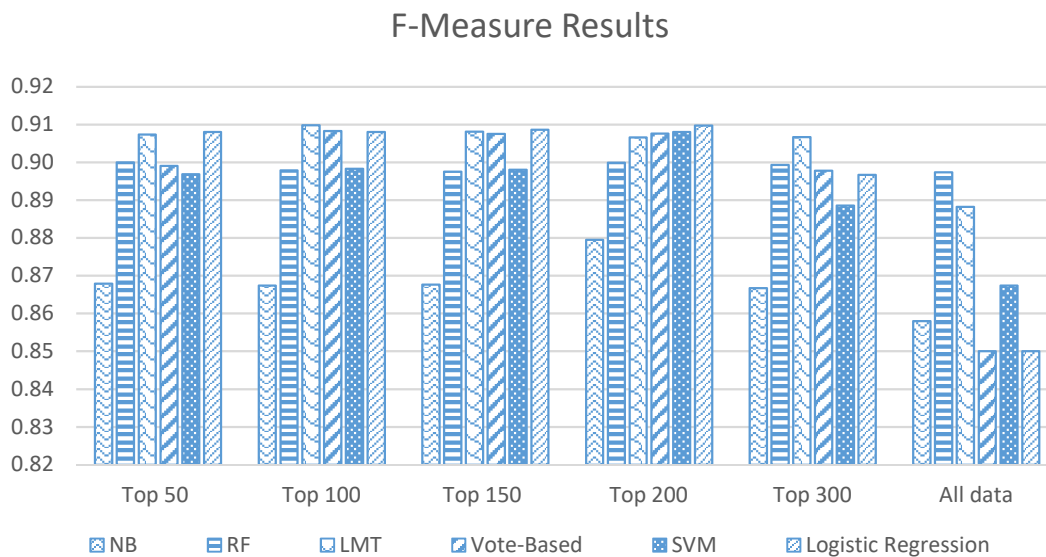


Figure 4: F-Measure Results of Sentiment-Based Machine Learning Models.

As observed in Figure 4, LMT sentiment-based model has shown steady performance with the best F-Measure value (91%) when the number of sentiment terms considered for severity prediction is 50, 100, 150, 200 and 300 (all top ‘N’ sentiment terms). Another observation from Figure 4 is that the RF model has a consistent F-Measure (90%) irrespective of the selected number of sentiment terms considered for predicting the severity level of bug reports.

As presented in Figure 5, LMT sentiment-based model has the best performance result with AUC value reaches 90% when the number of selected sentiment terms is 100, 150 and 200. The RF model also shows the best performance with AUC 88% when applied to the dataset that contains all sentiment features. However, RF performs less when compared to the performance of LMT. In contrast, the SVM model shows the least accuracy among other models. Furthermore, the performance of SVM

degrades as the number of selected sentiment terms considered for severity prediction increases. Nevertheless, other models indicate divergent performance results.

Generally, it can be observed from Figure 4 and Figure 5 that the performance results of all sentiment-based machine learning models are better when the IG feature selection method is exploited. The employed feature selection method indicates that bug reports severity can be predicted with outstanding performance. Specifically, the models perform very well when top-50 to top-300 sentiment terms are selected and considered by the severity prediction models compared to the results obtained when all sentiment features are considered. Thus, the models performance enhancement depends on the number of selected features used for severity prediction. When applying different machine learning algorithms on pre-processed datasets corresponding to top-50 until top-300 sentiment terms, it is clear from the results obtained that the best performance results, in terms of Accuracy, F-Measure and AUC are reported for top- 200 sentiment terms.

Based on the results reported above, the sentiment-based model generated by the LMT algorithm has shown the best performance results according to the F-Measure and AUC, with maximum performance results of 0.91 and 0.90, respectively. Therefore, the LMT model has effectively predicted the bugs severity level. The results also indicate that the RF and Logistic Regression models have achieved comparable results to the LMT algorithm. AUC and F-Measure maximum performance results for the RF model are 0.89 and 0.90, respectively and for Logistic Regression are 0.89 and 0.91, respectively. Furthermore, the Vote-Based model has the same attitude, but in certain aspects, less than the results observed from other models generated by the LMT, RF and Logistic Regression algorithms.

Thus, according to the analysis of the experimental results, the conclusion that can be drawn is that the machine learning model generated by the LMT algorithm outperforms other models for predicting the severity level of bug reports.

AUC Results

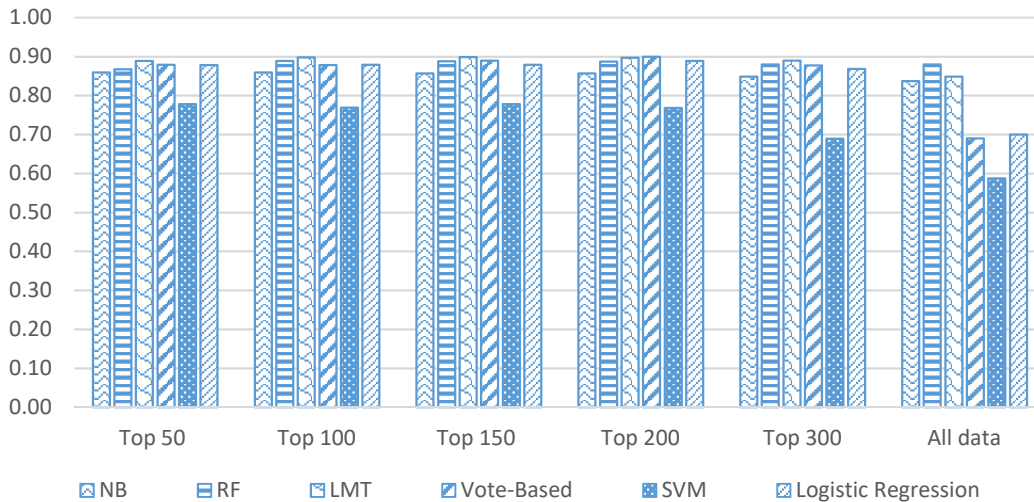


Figure 5: AUC Results of Sentiment-Based Machine Learning Models.

4.2 Lexicon-Based Approach Results

The overall number of unique sentiment terms extracted from the pre-processed dataset is 350, which forms 38% of the total distinct terms of 926 found in the dataset. Among 350 sentiment terms, 205 are positive and 145 are negative, which constitutes 59% and 41% of the total sentiment terms.

Table 3 below shows the distribution of positive and negative terms found in severe and non-severe bug reports acquired when performing sentiment analysis using SentiWordNet. As depicted in Table 3, the entire sentiment terms used to describe the non-severe bugs are 185, where 56% are positive and the remaining are negative terms. Whereas the total sentiment terms used to express the severe bug reports are 280, 42% are negative terms. From Table

3, it is observed that severe bug reports contain more positive than negative terms. This distribution affects the performance of the lexicon-based and machine learning approaches. In general, to obtain better results, the distribution of sentiment terms should be as follows: for severe bugs, the majority should be negative terms. While for non-severe bugs, the majority should be positive terms.

Table 3: Distribution of Sentiment Terms in Bug Reports

Sentiment Terms	Non-Severe	Severe
Positive Terms	104	163
Negative Terms	81	117

The lexicon-based model is built after the feature matrix is constructed based on the SentiWordNet dictionary according to equations 6, 7 and 8. As a result, the sentiment score of each bug report is calculated and its severity level is predicted. Then the results between actual and predicted severity for all bug reports are compared and reported. As depicted in Table 4, a confusion matrix is generated to demonstrate the performance results of the lexicon-based model.

Table 4: The Confusion Matrix of the Lexicon-Based Approach

		Actual		
		severe	Non-severe	Total
Predicted	severe	500	192	692
	Non-severe	351	121	472
	Total	851	313	1164

As observed from Table 4, among 851 bug reports classified as severe, the lexicon-based model has correctly predicted 500 severe bugs. The remaining 351 bug reports have not been predicted correctly as severe bugs. Whereas, among the 313 bug reports classified as non-severe, the lexicon-based model has correctly predicted 121 non-severe bugs and has failed to predict the remaining 192 bug reports correctly.

The results reported in the confusion matrix are used to calculate the performance metrics. Table 5 presents the evaluation metrics used to evaluate the performance of the lexicon-based model. These metrics are Accuracy, Precision, Recall (Sensitivity), Specificity and F-Measure.

As shown in Table 5, the lexicon-based model performs better in predicting severe bugs than predicting non-severe bugs. This observation is clear from the results of Recall (0.59) and Specificity (0.39). Further, it is observed from Table 5 that the

accuracy of the lexicon-based model is 0.53, which is low compared to the sentiment-based machine learning models.

Table 5: The Performance Results of the Lexicon-Based Approach

Performance Measure	Result
Accuracy	0.53
Precision	0.72
Recall	0.59
Specificity	0.39
F-Measure	0.65

The low performance of the lexicon-based approach can be attributed to how the software development team expresses their bug reports. When the software engineers detect bugs, they may not use proper and sufficient sentiment terms to describe the reported bugs, as evident by the limited number of sentiment terms (350 terms) found in all bug reports. Another reason that degrades the severity prediction performance is the incompatibility between the assigned severity level and the corresponding bug description in the bug reports summary field.

When comparing the performance results between the lexicon-based and sentiment-based machine learning approaches, it is apparent that the machine learning approach performs better than the lexicon-based approach in predicting the bug reports severity level. From the evaluation results mentioned above, the highest accuracy reached by the sentiment-based machine learning approach is 87.14%, while the lexicon-based approach accuracy reaches 53%. It is also clear from the results that the F-Measure of sentiment-based machine learning models with a maximum value of 0.91 is far superior to the lexicon-based results.

According to the analysis of the results, it can be concluded that the lexicon-based approach is not efficient for predicting the bug reports severity level present in the utilized closed-source dataset. However, the sentiment-based machine learning approaches have shown promising results and tremendous improvement for severity prediction on the same dataset. The severity prediction accuracy has been improved from 53% for the lexicon-based approach to 87.14% after utilizing the machine learning approach. Likewise, the F-Measure has been improved from 0.65 for the lexicon-based approach to 0.91 after utilizing the machine learning approach.

To reduce the gap between the performance of lexicon-based and sentiment-based machine learning approaches, the software engineers have to use more sentiment terms, either positive or negative terms, when describing bugs. The software engineers should also assign a proper severity level

consistent with what is expressed in the bug reports summary field.

5. THREATS TO VALIDITY

In this section, we introduce some of the potential threats to the validity of our study. The threat to construct validity is related to the performance metrics, which been selected for evaluation. Our study has exploited standard and well-known metrics used by many studies and researchers to evaluate the performance of the proposed sentiment-based models. These metrics are accuracy, Recall, precision, F-Measure and AUC measures.

Another threat to construct validity is the selection of a sentiment dictionary used for computing sentiment scores. In our study, we have employed the popular SentiWordNet lexicon for sentiment analysis. However, several sentiment lexicons are available specifically for software engineering text, such as SentiS4D [33], which may impact the performance of the sentiment-based approach.

The threat to internal validity is related to the implementation of our sentiment-based severity prediction approach. Even though we have verified our approach and checked the performance results to alleviate this threat, there could be some unobserved errors. Furthermore, the bug reports we use in this study belong to closed-source projects, reported and managed by the software development team of INTIX company. However, some of the bug reports severity levels may not be accurate, or the description of the bug reports are not appropriately written. Therefore, this threat may affect the results of severity prediction. Another threat could occur due to the imbalance distribution of the severity level of bug reports. Thus, the performance results of the sentiment-based models can be impacted.

The threat to external validity is the proposed sentiment-based severity prediction approach has been evaluated on a private dataset related to closed-source projects developed by a private Jordanian Company. In the future, in order to generalize the results, we plan to apply the proposed sentiment-based approach on other bug reports related to large open-source projects such as Mozilla, Eclipse and Netbeans to measure the effectiveness of the proposed methodology.

6. CONCLUSIONS

Many research works related to bug report severity prediction during the software maintenance phase have been proposed. Most of these studies have used traditional machine learning algorithms. However, few studies have considered and incorporated the bug reporters emotions in predicting the severity levels of bug reports. Thus,

the importance of this work lies in considering the emotions of bug reporters may improve the severity prediction accuracy.

In this paper, a sentiment-based methodology to predict the severity level of bug reports has been proposed. The proposed methodology has considered the reporters sentimental expressions present in the bug reports summary description. SentiWordNet dictionary has been used to identify the sentiment terms and compute their associated sentiment scores. A closed-source dataset extracted from the JIRA bug tracking system has been utilized to evaluate the proposed sentiment-based approaches.

Regarding the sentiment-based machine learning approach, five machine learning algorithms have been compared and evaluated. These algorithms are Naïve Bayes (NB), Logistic Regression, Vote-Based, Support Vector Machine (SVM), Random Forest (RF) and Logistic Model Tree (LMT). In addition, the sentiment-based machine learning models have been compared to the lexicon-based as a baseline approach. The results have shown that LMT outperforms all sentiment-based models, including the lexicon-based model.

According to the analysis of experimental results, it can be concluded that the lexicon-based approach has shown a low performance in severity prediction. Moreover, it is not efficient when it has been applied to the closed-source dataset. However, the sentiment-based machine learning approach has shown promising results, and the severity prediction performance is superior to the lexicon-based approach. From the reported experimental results, the sentiment-based machine learning approach has significantly improved the severity prediction accuracy to 87.14% compared to the baseline lexicon-based approach with an accuracy of 53%. Similarly, the sentiment-based machine learning approach has, to a large extent, improved the severity prediction F-Measure from 0.65 for the lexicon-based to 0.91.

One of the limitations of this work is that the proposed sentiment-based approach has been applied and evaluated on a dataset related to private closed-source projects. Therefore, to ensure the validity and the efficiency of our work, we intend to apply the proposed sentiment-based approach to other datasets related to open-source projects such as Mozilla and Eclipse. Furthermore, we plan to compare the proposed sentiment-based approach with similar research works that utilize similar open-source datasets.

Another limitation is that we have chosen a well-known sentiment lexicon (SentiWordNet) to predict the severity of bug reports. So, we will investigate whether using other sentiment lexicons such as Senti4SD and EmoTxt used in software engineering

text will improve the performance of the proposed sentiment-based approaches.

REFERENCES

- [1] Y. Tan, S. Xu, Z. Wang, T. Zhang, Z. Xu, and X. Luo, "Bug severity prediction using question-and-answer pairs from Stack Overflow," *Journal of Systems and Software*, p. 110567, 2020.
- [2] A. Chauhan and R. Kumar, "Bug Severity Classification Using Semantic Feature with Convolution Neural Network," in *Computing in Engineering and Technology*, ed: Springer, 2020, pp. 327-335.
- [3] Bugzilla. (2020, October). *Bugzilla Tracking System*. Available: <https://www.bugzilla.org/>
- [4] ATlassian. (2020, October). *JIRA Bug Tracking System*. Available: <https://www.atlassian.com/software/jira>
- [5] Q. Umer, H. Liu, and Y. Sultan, "Emotion based automated priority prediction for bug reports," *IEEE Access*, vol. 6, pp. 35743-35752, 2018.
- [6] H. M. Tran, S. T. Le, S. Van Nguyen, and P. T. Ho, "An Analysis of Software Bug Reports Using Machine Learning Techniques," *SN Computer Science*, vol. 1, p. 4, 2020.
- [7] L. A. F. Gomes, R. da Silva 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.
- [8] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *Journal of Systems and Software*, vol. 117, pp. 166-184, 2016.
- [9] T. Zhang, G. Yang, B. Lee, and A. T. Chan, "Predicting severity of bug report by mining bug repository with concept profile," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, pp. 1553-1558.
- [10] G. Yang, T. Zhang, and B. Lee, "Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports," in *2014 IEEE 38th Annual Computer Software and Applications Conference*, 2014, pp. 97-106.
- [11] A. Baarah, A. Aloqaily, Z. Salah, M. Zamzeer, and M. Sallam, "Machine Learning Approaches for Predicting the Severity Level of Software Bug Reports in Closed Source Projects," *Machine Learning*, vol. 10, 2019.
- [12] N. K. S. Roy and B. Rossi, "Towards an improvement of bug severity classification," in *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, 2014, pp. 269-276.
- [13] Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *2012 19th Working Conference on Reverse Engineering*, 2012, pp. 215-224.
- [14] K. Chaturvedi and V. 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 and Processes (IJOSSP)*, vol. 4, pp. 32-59, 2012.
- [15] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug," in *2011 15th European Conference on Software Maintenance and Reengineering*, 2011, pp. 249-258.
- [16] K. K. Sabor, M. Hamdaqa, and A. Hamou-Lhadj, "Automatic prediction of the severity of bugs using stack traces and categorical features," *Information and Software Technology*, vol. 123, p. 106205, 2020.
- [17] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *2008 IEEE International Conference on Software Maintenance*, 2008, pp. 346-355.
- [18] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, 2010, pp. 1-10.
- [19] A. Kaur and S. G. Jindal, "Text analytics based severity prediction of software bugs for apache projects," *International Journal of System Assurance Engineering and Management*, vol. 10, pp. 765-782, 2019.
- [20] G. Yang, K. Min, J.-W. Lee, and B. Lee, "Applying Topic Modeling and Similarity for Predicting Bug Severity in Cross Projects," *KSII Transactions on Internet & Information Systems*, vol. 13, 2019.
- [21] W. Y. Ramay, Q. Umer, X. C. Yin, C. Zhu, and I. Illahi, "Deep Neural Network-Based Severity Prediction of Bug Reports," *IEEE Access*, vol. 7, pp. 46846-46857, 2019.
- [22] G. Yang, T. Zhang, and B. Lee, "An emotion similarity based severity prediction of software bugs: A case study of open source projects,"

- IEICE TRANSACTIONS on Information and Systems*, vol. 101, pp. 2015-2026, 2018.
- [23] G. Yang, S. Baek, J.-W. Lee, and B. Lee, "Analyzing emotion words to predict severity of software bugs: A case study of open source projects," in *Proceedings of the Symposium on Applied Computing*, 2017, pp. 1280-1287.
- [24] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, pp. 130-137, 1980.
- [25] S. Baccianella, A. Esuli, and F. Sebastiani, "Sentiwordnet 3.0: an enhanced lexical resource for sentiment analysis and opinion mining," in *Lrec*, 2010, pp. 2200-2204.
- [26] K. P. Murphy, *Machine learning: a probabilistic perspective*: MIT press, 2012.
- [27] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5-32, 2001.
- [28] N. Landwehr, M. Hall, and E. Frank, "Logistic model trees," *Machine learning*, vol. 59, pp. 161-205, 2005.
- [29] A. Khan, B. Baharudin, L. H. Lee, and K. Khan, "A review of machine learning algorithms for text-documents classification," *Journal of advances in information technology*, vol. 1, pp. 4-20, 2010.
- [30] S. M. Weiss, N. Indurkha, T. Zhang, and F. Damerou, *Text mining: predictive methods for analyzing unstructured information*: Springer Science & Business Media, 2010.
- [31] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*: Morgan Kaufmann, 2016.
- [32] N. Japkowicz and M. Shah, *Evaluating learning algorithms: a classification perspective*: Cambridge University Press, 2011.
- [33] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli, "Sentiment polarity detection for software development," *Empirical Software Engineering*, vol. 23, pp. 1352-1382, 2018.