Machine Learning Approaches for Predicting the Severity Level of Software Bug Reports in Closed Source Projects

Aladdin Baarah[1], Ahmad Aloqaily[2], Zaher Salah[3], Mannam Zamzeer[4], Mohammad Sallam[5]
Department of Software Engineering, Hashemite University, Zarqa, Jordan[1, 5]
Department of Computer Science and its Applications, Hashemite University, Zarqa, Jordan[2]
Department of Computer Information Systems, Hashemite University, Zarqa, Jordan[3]
Department of Information Technology, the University of Jordan, Amman, Jordan[4]

***Abstract*—In Software Development Life Cycle, fixing defect bugs is one of the essential activities of the software maintenance phase. Bug severity indicates how major or minor the bug impacts on the execution of the system and how rapidly the developer should fix it. Triaging a vast amount of new bugs submitted to the software bug repositories is a cumbersome and time-consuming process. Manual triage might lead to a mistake in assigning the appropriate severity level for each bug. As a consequence, a delay for fixing severe software bugs will take place. However, the whole process of assigning the severity level for bug reports should be automated. In this paper, we aim to build prediction models that will be utilized to determine the class of the severity (severe or non-severe) of the reported bug. To validate our approach, we have constructed a dataset from historical bug reports stored in JIRA bug tracking system. These bug reports are related to different closed-source projects developed by INTIX Company located in Amman, Jordan. We compare eight popular machine learning algorithms, namely Naive Bayes, Naive Bayes Multinomial, Support Vector Machine, Decision Tree (J48), Random Forest, Logistic Model Trees, Decision Rules (JRip) and K-Nearest Neighbor in terms of accuracy, F-measure and Area Under the Curve (AUC). According to the experimental results, a Decision Tree algorithm called Logistic Model Trees achieved better performance compared to other machine learning algorithms in terms of Accuracy, AUC and F-measure with values of 86.31, 0.90 and 0.91, respectively.**

Keywords*—Software engineering; software maintenance; bug tracking system; bug severity; data mining; machine learning; severity prediction; closed-source projects

I. Introduction

Over the past years, the process of fixing bugs in the software maintenance phase has become a challenging task because the number of reported software bugs in large software systems (e.g., open-source projects) is growing massively [1]. For instance, the number of reported bugs submitted daily to Mozilla open-source system are 135 on average [2]. Managing a high volume of new bugs submitted daily by different reporters in these large open systems is a difficult task. Furthermore, this increases the amount of work done by the so-called bug triager, who assesses and analyzes these bugs within the bounds of time and resources to assign an appropriate severity level and suitable developer(s) who will fix those defected bugs [3].

Bug report systems are important software artifacts. They are leveraged for various tasks during software maintenance, such as assigning bugs to appropriate developers, assessing the severity and priority of bugs and detecting duplicate bugs. The quality of bug reports relies, to a large extent, on the information written in these reports [4]. Thus, if the data in bug reports are incomplete, unclear or inaccurate, the tasks as mentioned above will lead to unpredictable results.

Bug tracking systems such as Bugzilla [5] and JIRA [6] are utilized by open-source and closed-source projects during the software maintenance phase to collect, maintain, manage, and track issues related generally to bug reports (i.e. corrective maintenance) [7]. At present, developers, quality assurance testers, users and other team members are promoted to report and submit bugs they experience to bug repositories in a situation when the project behaves incorrectly and does not conform to the software requirements [8].

Bug severity "is the degree of impact that a defect has on the development or operation of a component or a system" [9]. Generally, bug reports are categorized according to their severity. High severity reports exemplify major (i.e. critical or fatal) errors and have a high impact on the functionality of the system, as well, they are given more top priority and should be resolved rapidly. While low severity reports exemplify trivial errors and minor problems that do not affect the execution of the system [10].

Initially, when a new bug is created, the reporter (e.g., developer) is required to fill in the fields of the bug report form such as a one-line summary for a specific project. As well, the reporter is required to estimate the severity level field (e.g., high, medium and low) of the observed bug according to their competence and knowledge. In practice, if the reporter is incapable of assessing the severity of the bug, they will assign a default value for the severity field and thus makes the triaging process more difficult [11]. One potential explanation for assigning a default value for the severity is that the reporters are unable to distinguish between different severity levels, or they have lack of knowledge and experience in assigning the value of the severity level [12]. Another potential

explanation is that the reporters do not take into consideration the severity estimation at all when they submit a report and leave the default value unchanged [10].

Even though there are rules on how to set the severity label of the encountered bugs, specifying proper severity level is mostly based on the expertise of who reports the bug. The developer has to determine the severity of these software bugs manually to prioritize which bug report needs more attention than others in terms of fixing. In the case of assigning an inaccurate value for the severity, this will result in postponement in fixing severe bugs, and this will expand the time to resolve these type of bugs [13]. However, a further validation step is required after the bug has been submitted where the bug triager has to confirm the validity of the severity value, whether it is adequately assigned or not. This process is called "severity identification" [14]. It is a tiresome and time-consuming process and it increases the volume of work carried out by the triager especially when there is a massive number of bugs submitted daily to the bug tracking systems.

Fig. 1 shows an example of the JIRA bug report used by INTIX Company. This bug report is reported by "Mohammad Yasser" and assigned to the developer "Mustafa Alqudah". The one-line summary "Incorrect unit price in order details if a product has multi-selection attribute" is shown in the top left of the figure and the severity level assigned for this bug is high.

To overcome the mentioned problems, there is a necessity to automate the whole process of assigning the severity level of newly reported bugs to replace the manual job. One reason for that is to decrease the amount of work done by the triager. Another reason is to improve the accuracy of severity identification.

In this paper, we compare different machine learning algorithms applied on the textual description of the bug reports (i.e. one-line summary field). Unlike most of the research works reported in the literature, we applied the proposed methodology to a private dataset related to bug reports. These bugs are associated with closed-source projects developed by INTIX Company located in Amman, Jordan. The dataset is extracted from JIRA bug tracking system used by the company.

The structure of this paper is organized as follow. In Section II, we describe the related works conducted by other researchers. Then, in Section III, we demonstrate our research methodology in detail. After that, in Section IV, we discuss the experimental results. Finally, we summarize our approach and point out future work in the conclusion section.

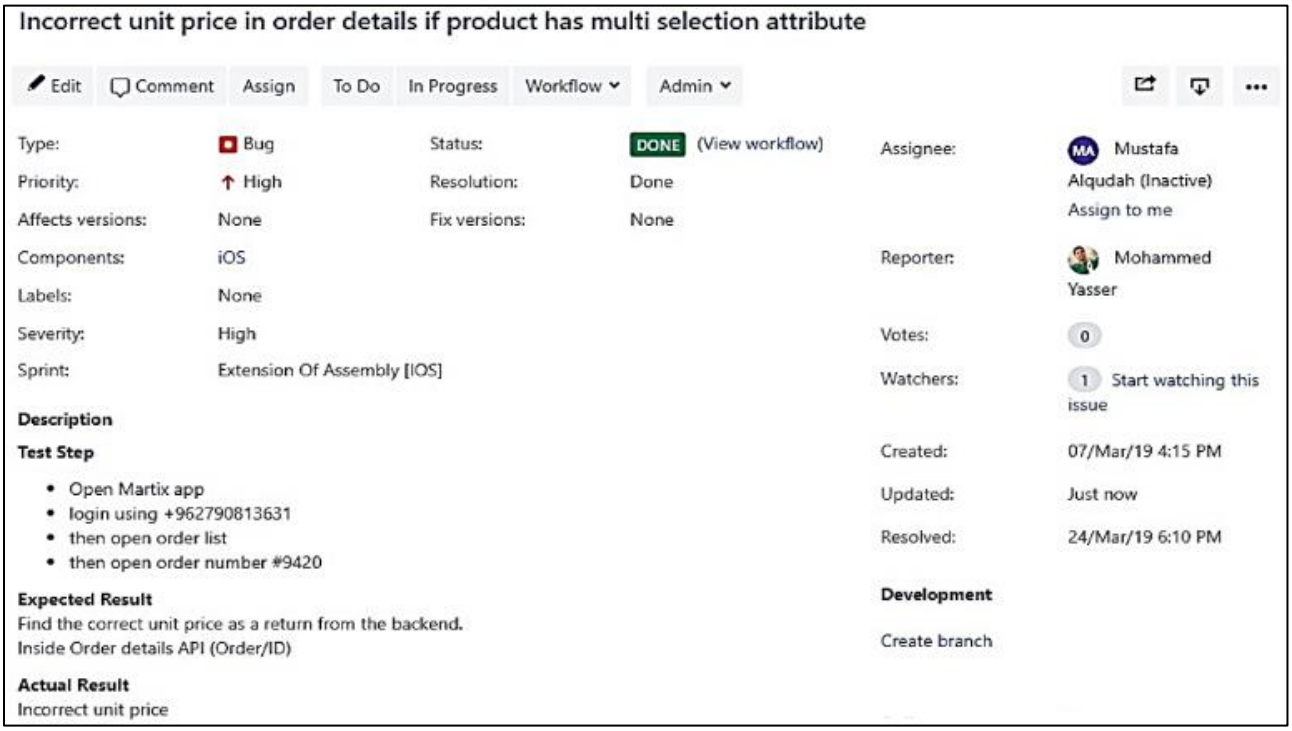

Fig. 1. Example of JIRA Bug report.

II. Related Works

The work described in this paper is concerned mainly with bug reports, particularly in the area of bug severity. This section presents the recent literature review regarding bug severity prediction.

One of the first studies to predict the severity label of bug reports was performed by [15]. A rule-based learning technique was utilized to build a new tool called SEVERIS. SEVERIS is based on text mining and machine learning techniques applied on the unstructured data of the bug report (i.e. summary, description). They applied their automated prediction model on NASA's Project and Issue Tracking System (PITS). The authors argued that SEVERIS, with a slight adjustment, can be applied to other open-source repositories such as Bugzilla.

An extension of [15] work was performed by [11] to investigate whether text mining techniques applied in the textual information of a bug report is an adequate approach to predict the severity of a newly reported bug automatically. They used Naïve Bayes technique for their prediction model and they applied their model on datasets extracted from three popular open-source bug repositories, including Mozilla, Eclipse and GNOME. The experimental results showed that prediction accuracy was varied between 0.65–0.75 for Mozilla and Eclipse dataset. Regarding GNOME dataset, the prediction accuracy was varied between 0.70–0.85.

Later, a follow-up study was conducted by [10]. The authors compared four familiar classification techniques (Naïve Bayes, Naïve Bayes Multinomial (NBM), K-Nearest Neighbor (K-NN) and Support Vector Machine (SVM)) to figure out which technique was the most suitable for the severity prediction model so that it can classify the newly reported bug into severe and non-severe. For evaluation purposes, they construct twelve different datasets from Eclipse and GNOME open-source projects. According to their experimental results, among the four candidate classifiers, NBM had the best results in terms of accuracy.

A comparable study to [15] was conducted by [16]. The authors conducted different experiments on four nameless datasets of NASA's PITS using three methods: a Regression method called Multi-Nomial Multivariate Logistic Regression (MMLR), Multi-Layer Perception (MLP) and Decision Tree. The prediction models were fed with different top-k terms extracted from the training datasets using Information Gain (IG) feature selection. The authors concluded that the Decision Tree performed better than MMLR and MLP.

A further study for predicting fine-grained severity level of a bug report was conducted by [17]. The authors built a model to classify the severity of a new bug reported to the bug repository. A dataset of 163 bug reports was built from Eclipse and Mozilla projects. Six machine learning approaches were adopted, namely, Naïve Bayes, RBF Networks, Functional Trees, Random Trees, Random Forests and AdaBoost. In accordance with their results, AdaBoost with base classifiers, as mentioned above, showed an improvement of up to 4.9% in terms of accuracy.

Other studies, for example [18, 19], examined the impact of utilizing other attributes of bug reports, besides the unstructured text, to enhance the bug severity prediction. In the study conducted by [18], the authors used NBM as a classification technique and applied their proposed approach on two distinct datasets originating from two open-source projects, namely Mozilla and Eclipse. Their empirical study revealed that there was an improvement in the prediction accuracy and outperformed the work of [11]. Similarly, Yang et al. [19] adopted NBM classifier for severity prediction. According to their evaluation, the attributes (quality indicators) of bug reports from Eclipse dataset (i.e. stack traces, attachments) exploited in their study showed an improvement in the accuracy of their prediction model.

In the domain of cross-project severity prediction, Singh et al. [20] conducted a study based on the summary description of the bug report. Different prediction models were built using different classifiers, namely SVM, K-NN and Naïve Bayes. A combination of seven datasets from seven Eclipse projects was constructed to build 63 training set candidates. The experimental results revealed that K-NN achieved better results than SVM and Naïve Bayes when using a combination of several training datasets rather than a single training dataset.

Many research works, as described in [21-25] employed different feature selection methods to minimize the number of informative features set and to enhance the accuracy of the classifier. All the works mentioned above paid particular attention to the effectiveness of feature selection techniques on the accuracy of classifying the severity of bug reports. Bi-gram, along with feature selection and text mining algorithms were proposed by [22] to enhance the accuracy of their prediction model. While in work described by [25], the authors examined three feature selection methods, namely, Information Gain (IG), Chi-Square (CHI) and Correlation Coefficient to extract the distinct features that depict the severity of the bug reports (severe or non-severe) from the summary field in the bug reports.

On the other hand, an ensemble feature selection technique was proposed by [21] through consolidating at most two feature selection methods. Various feature selection methods were used in their experiments, in particular, Term Frequency (TF), Document Frequency (DF), Mutual Information (MI), Statistical Dependency (SD), Information Gain (IG), Chi-Square (CHI) and Correlation Coefficient. In their prediction model, the authors employed Naïve Bayes Multinomial as the classifier.

Later, [26] proposed a new approach, an integration of topic modelling using LDA and similarity using KL-divergence to predict the severity of bug reports in cross projects. In their work, a total of 20,000 bug reports from 4 different open source projects (Eclipse, Mozilla, WireShark and Xamarin) were assembled to validate their proposed approach. The experimental results demonstrated that their model achieved better performance, in terms of accuracy than other four cutting-edge studies listed in their literature.

In more recent studies, emotional-based expressions written in the unstructured text fields of the bug reports were leveraged by [27, 28] to classify the severity label of bug report (i.e., critical, high, trivial and low). In the former approach [27], two stages related to emotional similarity technique were performed using Smoothed Unigram Model and KL-Divergence. In the latter approach [28], the authors exploited the notion of deep learning algorithm based on Neural Network along with emotional analysis. The experimental results of both studies above emphasized that employing emotion analysis significantly improved the performance of the prediction model.

In the context of bug report severity prediction, most of the reported research studies in the literature mainly used open-source projects because they are publicly accessible. While in the research work described in this paper, we are attempting to exploit a private bug report dataset related to closed-source projects developed by an existing local company located in Jordan.

III. Research Methodology

The process of predicting the severity label of reported bugs is shown in Fig. 2. It comprises of four main phases: dataset extraction, dataset pre-processing, feature selection and prediction. The following sections will explain each phase in detail.

A. Dataset Extraction

The bug reports dataset is extracted from the repository of JIRA bug tracking system related to closed-source projects developed by INTIX Company located in Amman, Jordan. We considered bug reports submitted to JIRA between May 2016 and March 2018.

The bug reports were classified into five levels: lowest, low, medium, high and highest. Although software engineers usually follow a specific guideline on how to assign severity of reported bugs, however, the categorization process seems to be evaluated imprecisely. In this research work, we treat lowest and low severity as non-severe, while high and highest severity as severe bugs. Furthermore, as proposed by [12], the authors recommended not to take the medium severity in the classification process. The medium class, as it is investigated, is the default option for reporting a bug and it seems that a large number of reporters report any confusing bug as a medium.

The datasets mainly consist of three features including bug ID, a short description (i.e., one-line summary) and the severity level of the bug. The description of each bug is represented as a short text (snippet). From the bug reports, the severity and the short description of the bug report were mainly used for the prediction process. Table I statistically summarizes the characteristics of the dataset used in the experiments described in this research work. As shown in Table I, too many words are repetitive in the dataset where the number of distinct words is significantly fewer than the total number of all words exists within the dataset (765 out of 9016). The table also shows the shortness property of the bug summary (i.e. short text) where the average number of words per bug summary was 7.75 words and thus this made the prediction process more challenging.

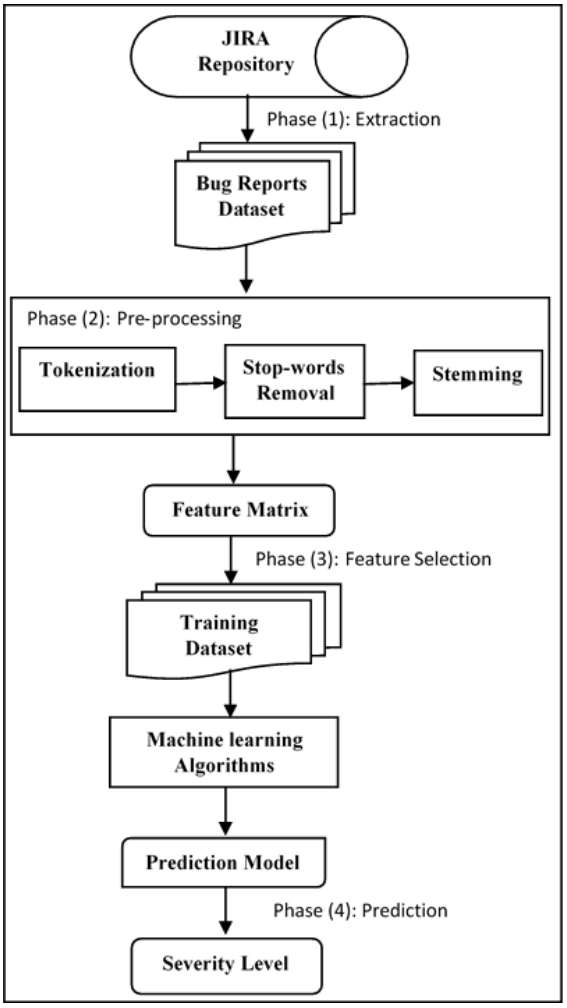

Fig. 2. Methodology for Predicting Bug Severity.

TABLE. I. Statistical Summary for the Bug Reports in JIRA Dataset

Min number of words per Bug Summary	1
Max number of words per Bug Summary	44
Avg. number of words per Bug Summary	7.75
StD. number of words per Bug Summary	3.73
Total Number of Bug Reports in the dataset	1164
Number of Severe Bugs	851
Number of Non-Severe Bugs	313
Number of all words in the dataset	9016
Number of distinct words in the dataset	765

B. Dataset Pre-Processing

To build a prediction model, machine learning algorithms require text data (i.e., one-line summary) of a bug report to be transformed into a feature of vector (Bag-of-Words). Therefore, different pre-processing steps must be applied to the dataset (text) to be converted into a vector of features where the features represent words in a utilized dataset. The typical pre-processing phase starts with Tokenization, Stop-words Removal and Stemming. These steps are explained as follow:

1) Tokenization: This step represents splitting a text data into a collection of words where each word corresponds to a single term. Removing white spaces, punctuations and converting all uppercase characters to lowercase are taken into consideration in this step.

2) Stop-words removals: In the domain of natural language processing, conjunctions, adverbs, prepositions and other constructive terms are used mainly to build a sentence. These terms do not carry out any semantic or statistical information to distinguish the text. Terms like "an", "on", "there" and many others are called stop-words and are not crucial for bug severity prediction. Therefore, all stop-words are removed based on a pre-defined list of stop-words.

3) Stemming: Stemming is one of the main steps in the domain of text mining and natural language processing. The purpose of carrying out this step is to replace all terms that have a common stem (root word) and the stem of a word is retained as a feature. For instance, the words "find", "finds", "found" and "finding" can be replaced with one word or their stem which is "find".

Finally, all the terms obtained after the three aforementioned pre-processing steps are called features or bag-of-words and are mainly used to build prediction models as described later in this section.

C. Feature Selection

The size of the feature set that can be generated after applying pre-processing steps is still enormous and consequently is not suitable for machine learning algorithms. Feature selection, in text mining, is a crucial step to select the most discriminative features (terms/words) from a list of features using appropriate feature selection methods.

There are many available feature selection methods in the literature such as Chi-square, Information Gain (IG), Term Frequency, Document Frequency, and Mutual Information. In fact, there is no much difference in utilizing different feature selection methods as we decide to select sets of features with different sizes [29, 30]. For experimental purpose, IG feature selection method is applied and features obtained after pre-processing are ranked and the top 'N' scoring features are selected based on ranking results to build the prediction models of different datasets.

IG feature selection is used to measure the dependency between features and the class label [31]. It measures the informativeness of a feature gained regarding the class label and is defined as follows:

$$IG(f_i, c_j) = H(f_i) - H(f_i|c_j)$$

where f_i is the feature (term) i and c_j is the class label j, $H(f_i)$ the entropy of term f_i, and $H(f_i|c_i)$ is the entropy of f_i after observing class label c_j. The entropy $H(f_i)$ is defined as

$$Entropy = -\sum_i p_i \log_2 p_i$$

Entropy is a common way to measure the degree of randomness or impurity of a variable and comes from information theory domain. The higher the entropy of a variable the more the information it holds about the class.

Finally, the bug reports are represented as a feature matrix where each row represents a bug report consisting of n selected features (terms). Each term is weighted using the Term Frequency Inverse Document Frequency (TF-IDF) approach. Term frequency is calculated by multiplying term frequency with inverse document frequency and is given as:

$$\text{TF-IDF} = n_w^d \log_2 \left(\frac{N}{N_w}\right)$$

Where n_w^d the frequency of word w in document d, N is the number of document and N_w are documents containing word w. The *TF-IDF* value indicates that words (terms) which occur frequently in a specific document are more significant than other terms in the same document. Lastly, the features are normalized.

D. Machine Learning Algorithms

In this section, different machine learning algorithms for predicting the severity level of the bugs are described and utilized in this study. These algorithms are mainly concerned to deal with unstructured data such as text [32]. In this study, machine learning algorithms namely: Bayesian algorithm, Support vector machine, decision tree and rule-based algorithms are used to classify severity levels of bug reports. These algorithms are described in the following subsections:

1) Bayesian algorithms: Bayesian is a probability-based classification algorithm that is based on Bayesian theorem [33]. It describes the probability of an independent variable based on prior knowledge of the dependent variable(s). In this research study, two different variations of the Bayesian algorithm were used namely, Naïve Bayes classifier (NB) and Naïve Bayes Multinomial (NBM).

The Naïve Bayes classifier estimates the class conditional probability of class label with the assumption that all attributes are independent [34]. The NB has been widely used in the domain of text classification due to its simplicity and effectiveness. On the other hand, NBM is similar to the naïve Bayes classifier except that it considers a weight for each feature during probability calculations. The weight of each feature can be determined by specific distributions or as a parametric model. The parameters can be estimated based on the training data. For the case of NBM, the distribution of data is assumed as a multinomial model [35].

2) Support vector machine: The main idea underlying support vector machines (SVMs) is as follow: search the best maximal margin hyperplane separating two classes of a given training data. A margin is defined as the sum of the distances of the closest positive and negative correctly classified data points (support vectors) from the hyperplane while penalizing misclassified data points. In the case of linear classification problems, linear SVMs can be used to search for the

hyperplane in the original space. On the other hand, for nonlinearly separable issues, the data are implicitly mapped to a higher dimensional space through a kernel function, where non-linear SVMs can be used to find a separating hyperplane.

For any given classification problem, if no hyperplane can separate the two classes, a soft margin approach can be used to control the sensitivity of outliers to allow a separating hyperplane. It determines a hyperplane that splits the cases as clearly as possible with a penalty for misclassified cases, while still maximizing the distance to the nearest support vectors. SVMs have been widely used in applications such as handwriting recognition and bioinformatics and showed superior performance [36-38]. Careful design and methodological approach must be taken in applying SVM algorithms including tuning of parameters. SVMs has also been applied in the domain of text mining as it shows satisfactory results. The high dimensionality of text data makes SVMs a useful algorithm to apply and also avoids the curse of dimensionality problem of text data [39]. An implementation of SVMs named SMO algorithm [40] has been applied to the dataset, which is an open-source implementation of SVMs.

3) Decision tree: The decision tree (DT) is a machine learning predictive approach used for classification and regression problems. It creates a prediction model by learning decision rules from data on a tree-like model. A DT model has a tree structure in which each node represents a test on a given variable and each branch represents the outcome of that test. Leaf nodes of the tree represent a class label (decision). The path from the root to leaf nodes represents a classification rule. Different DT based algorithms have been implemented including J48, random tree, random forest, Logistic model trees, and many others. Generally, decision tree classifiers have good accuracy. It is a typical inductive approach to learn knowledge from data.

In this study, three DT algorithms were applied. These algorithms include J48 [41], Radom Forest (RF) [42], and Logistic Model Trees (LMT) [43].

4) Rule-based algorithms: The Rule-Based Algorithm extract knowledge from data in the form of rules. Rules usually take the form of an if-then expression. The main feature of rule-based models is that the produced model is expressed in term of a set of rules rather than just one rule or model. In this study, an algorithm called (Repeated Incremental Pruning to Produce Error Reduction (RIPPER) [44] is applied to the dataset. The RIPPER algorithm works through sequentially runs over the data in multiple passes. This algorithm repeatedly learns one rule at each pass until no data left. The JRip implementation of the RIPPER algorithm in WEKA is applied to the utilized dataset.

To conclude, the classification algorithms namely: NB, NBM, SVMs, J48, RF, LMT, JRip and KNN are mainly utilized to evaluate their performance on the studied dataset. The open-source Weka software [45] was used to run the experiments with different parameters setting and the best parameters setting are selected based on evaluation measures.

E. Experimental Settings

The validation approach used in our study to evaluate the performance of different classifiers is the k-fold cross-validation approach. This approach was used to generate a test set and avoid the over-fitting problem. The cross-validation approach performs independent tests without requiring separate test data samples and without reducing the data samples used to build prediction models. In the k-fold cross-validation process, the original data is classified into k groups, so that each group is used once as a validation set and the remaining data as a training set. In this study, the 10-fold cross-validation was used to evaluate the performance of different classifiers. Stratified based sampling was used to generate training and testing sets. In each case, a prediction model was trained using 9 folds of the data and tested on the remaining fold and the results were averaged.

F. Performance Evaluation

Once a classifier is trained successfully and a prediction model is generated, performance evaluation measures must be applied on a separate dataset called test set to evaluate the performance of the generated model. In our experiments, the performance of the generated prediction models is evaluated using various performance measures; these measures include accuracy, F-measure and Area Under the Curve (AUC). Each measure provides a different perspective of performance evaluation and a broader set of performance results to compare.

The accuracy measures how correctly a classifier predicts class labels. It is calculated as the percentage of true positive and true negative rates to the number of all instance. On the other hand, the F-measure considers both the precision and the recall to compute the performance of a classifier and is measured as the harmonic mean of precision and recall. In this case study, the *F1* measure is used, where recall and precision are equally weighted:

$$\text{F-measure} = 2 * \frac{precision * recall}{(precision + recall)}$$

The F-measure reach a value of 1 (perfect precision and recall) and the worst value is 0. The higher F-measure value indicates the higher quality performance of the classifier.

Furthermore, an alternative measure to evaluate the performance of generated models is the Receiver Operating Characteristic (ROC). This approach compares the true positive rate with a false positive rate as a drawn curve. The ROC measure is usually summarized as a statistical value representing the area under the ROC carve known as Area Under Curve (AUC). The AUC represents the probability that the outcome of the generated model is better than induction using a random model, where a random model has an AUC value of 0.5 while a perfect model has an AUC of 1. Therefore, the higher AUC value is, the better the model achieves.

IV. Experimental Results and Discussions

In this research study, eight machine learning algorithms were employed on the studied dataset (JIRA bug reports). These algorithms are NB, NBM, SVMs, J48, RF, LMT, JRip and KNN. As discussed before, IG feature selection was applied and features obtained after pre-processing are ranked

and the top 'N' scoring features are selected based on ranking results. A set of eight datasets are generated, these datasets include the original dataset with the whole feature set and the top 25, 50, 75, 100, 125, 150, 200 and 300 terms are generated as new datasets to build prediction models. The results then analyzed in terms of selected performance measures, which are the accuracy, F-measure and AUC measures. These measures are used to examine the performance of machine learning algorithms empirically and the best models were reported. The open-source Weka software [45] was used to run the experiments with different parameters setting and the best parameters setting are selected based on evaluation measures.

Table II shows the results of different machine learning algorithms concerning the original JIRA dataset with all feature set generated after pre-processing. Based on the accuracy, it is found that the accuracy of different classifier varies in the range of 75.23% and 84.55%. However, the accuracy results are somehow biased as the dataset is imbalanced and the results are biased toward the larger class. The reported performance results of all algorithms based on the AUC and F-measure are promising as these measures take into consideration the performance results of all classes. As seen in Table II, the AUC performance results vary between 0.59-0.88 and F-measure between 0.85-0.90. These measures are not biased and report the performance of the two classes. Based on the F-measure performance results, all DT algorithms, namely Random Forest, LMT and J48 perform the best followed by NBM and JRip and SVM, and the KNN performed the lowest.

Table III shows the performance of different machine learning algorithms on the studied dataset with a varying number of selected features based on the IG ranging from 25-300 features. As shown in Table III, The performance results based on all measures show comparable and better results than the ones obtained from the generated models on the original dataset (with no feature selection). Therefore, the selected terms based on feature selection will able to distinguish the bug severity based on a smaller feature set. The accuracy reaches 86.54% when the number of selected features is 75 features, as compared to the model generated based on the original dataset (with all features) which was 84.55%. In terms of the AUC measure, the AUC result reaches 0.91 when the number of selected features is 125 features, as compared to the model generated based on all features which was 0.88. Furthermore, the F-measure results reached 0.91 when the number of selected features is 125 features, as compared to the model generated based on all features which was 0.9.

Fig. 3 shows the performance results of all machine learning algorithms, in terms of accuracy, on varying generated datasets based on the utilized feature selection method. According to the results, it is found that the accuracy of most algorithms stabilized when the number of selected features is 75 or more terms except NB classifier. The accuracy of NB algorithm has consistent accuracy in all datasets with a varying number of features. The performance results of AUC and F-Measure of various algorithms are shown in Fig. 4 and Fig. 5 From these figures, the performance of all classifier depends on the number of terms considered to build the classifier and the performance results provide comparable and better results than relying on the whole feature set. The reported results of the AUC and F-measure using different machine learning algorithms state that the best performance results are when the number of selected features is 125 terms. These terms are chosen using the information gain measure. It is clear that the severity of bug reports can be predicted with a reasonable performance of AUC measure as they show a differentiated and progressive value of AUC and better than using whole feature sets. Similar trends are also observed with the F-measure shown in Fig. 5 From results reported in Table III and Fig. 5, the best F-measure results were obtained using Naïve Bayes Multinomial, SVM, and Logistic model trees. These algorithms receive 90%-91% performance results where Logistic model trees achieve best and stable results. However, other algorithms report varied and fewer performances.

Similar results observed from Table III when the number of terms taken into account is over 75 terms. The AUC and F-measure results are similar and consistent as compared to the performance results when the number of selected terms are less. The results suggest that the generated models can predict the severity of bugs more accurately when the number of selected terms in the range of 75 to 125 terms. As Table III shows, it can be concluded that models have performed well in predicting the bug reports of either severity levels as reported by both AUC and f-measures values.

TABLE. II. PERFORMANCE RESULTS OF DIFFERENT CLASSIFICATION ALGORITHMS ON JIRA DATASET WITH ALL FEATURES

Classifier	Accuracy	AUC	F-measure
SVM	77.5	0.59	0.87
KNN	75.23	0.75	0.85
DT - J48	80.48	0.76	0.87
DT - RF	84.55	0.88	0.9
Rule-Based JRip	81.28	0.72	0.88
NBM	82.43	0.86	0.88
Naïve Bayes	80.03	0.84	0.86
DT:LMT	83.26	0.86	0.89

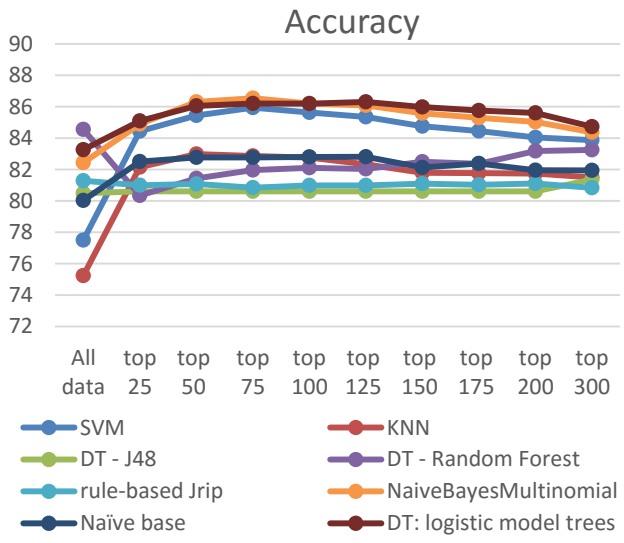

Fig. 3. Accuracy of different Classification Algorithms.

TABLE. III. ACCURACY OF DIFFERENT CLASSIFICATION ALGORITHMS WITH A VARYING NUMBER OF SELECTED FEATURES BASED ON IG

	Top 25			*Top 50*			*Top 75*		
	Accuracy	*AUC*	*F-measure*	*Accuracy*	*AUC*	*F-measure*	*Accuracy*	*AUC*	*F-measure*
SVM	84.44	0.75	0.90	85.44	0.77	0.91	85.94	0.78	0.91
KNN	82.15	0.83	0.88	82.98	0.84	0.89	82.86	0.85	0.89
DT - J48	80.61	0.74	0.88	80.61	0.74	0.88	80.61	0.74	0.88
DT - RF	80.34	0.84	0.87	81.43	0.85	0.87	81.96	0.86	0.88
rule-based JRip	80.99	0.72	0.88	81.08	0.72	0.88	80.84	0.72	0.88
NBM	84.89	0.86	0.90	86.31	0.90	0.91	86.54	0.91	0.91
Naïve Bayes	82.50	0.86	0.88	82.77	0.87	0.88	82.77	0.87	0.88
DT:LMT	85.09	0.87	0.90	86.05	0.89	0.91	86.20	0.90	0.91

	Top 100			*Top 125*			*Top 150*		
	Accuracy	*AUC*	*F-measure*	*Accuracy*	*AUC*	*F-measure*	*Accuracy*	*AUC*	*F-measure*
SVM	85.63	0.77	0.91	85.35	0.76	0.91	84.76	0.75	0.90
KNN	82.76	0.84	0.89	82.33	0.85	0.88	81.79	0.84	0.88
DT - J48	80.61	0.74	0.88	80.61	0.74	0.88	80.61	0.74	0.88
DT - RF	82.11	0.86	0.88	82.03	0.87	0.88	82.49	0.87	0.88
rule-based JRip	80.98	0.72	0.88	80.98	0.72	0.88	81.10	0.72	0.88
NBM	86.20	0.91	0.91	86.08	0.90	0.91	85.58	0.90	0.91
Naïve Bayes	82.80	0.87	0.88	82.81	0.87	0.88	82.13	0.86	0.88
DT:LMT	86.19	0.90	0.91	86.31	0.90	0.91	85.98	0.89	0.91

	Top 175			*Top 200*			*Top 300*		
	Accuracy	*AUC*	*F-measure*	*Accuracy*	*AUC*	*F-measure*	*Accuracy*	*AUC*	*F- measure*
SVM	84.45	0.74	0.90	84.04	0.73	0.90	83.87	0.72	0.90
KNN	81.77	0.84	0.88	81.74	0.83	0.88	81.49	0.81	0.88
DT - J48	80.61	0.74	0.88	80.61	0.74	0.88	81.36	0.78	0.88
DT - RF	82.36	0.87	0.88	83.18	0.87	0.89	83.25	0.87	0.89
rule-based JRip	81.02	0.72	0.88	81.11	0.72	0.88	80.83	0.72	0.88
NBM	85.30	0.90	0.90	85.03	0.90	0.90	84.37	0.88	0.90
Naïve Bayes	82.38	0.86	0.88	81.95	0.86	0.88	81.94	0.86	0.88
DT: LMT	85.75	0.89	0.91	85.60	0.89	0.91	84.73	0.88	0.90

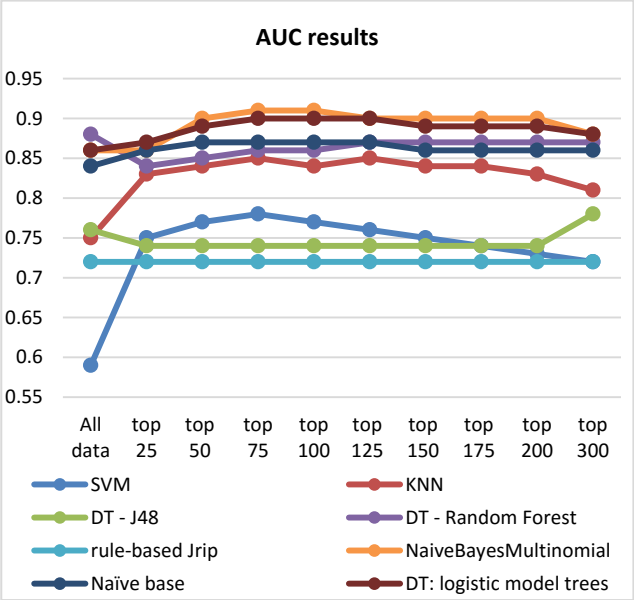

Fig. 4. The AUC Results of different Classification Algorithms with a Varying Number of Selected Terms.

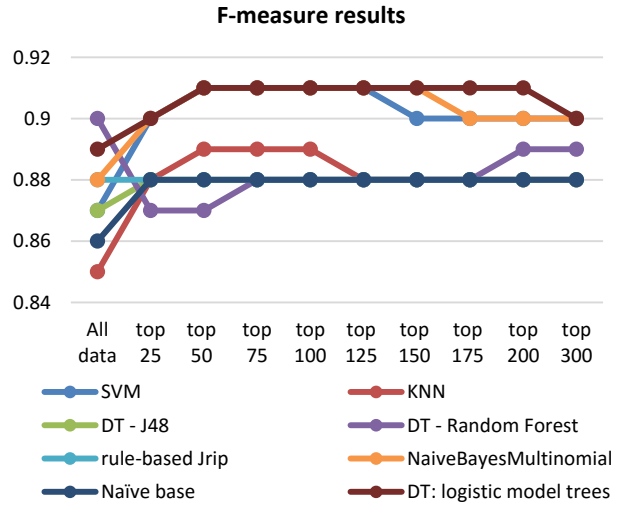

Fig. 5. The F-Measure Results of different Classification Algorithms with a Varying Number of Selected Terms.

Overall, the LMT algorithm reported the best performance results based on all performance measures. The maximum AUC and F-measure obtained with LMT are 0.90 and 0.91, respectively. It is further found that NBM performs similarly to LMT in terms of AUC and F-measure. The minimum AUC and F-measure values obtained with NBM are 0.88 and 0.91 respectively, and the maximums are 0.9 and 0.91, respectively. The SVM model also has comparable performance results, but it is in some ways less than the results reported for the LMT and NBM. The maximum AUC and F-measure results obtained with SVM are 0.8 and 0.9, respectively. Other algorithms report divergent accuracy performance results. The maximum AUC and F-measure results obtained with other algorithms are 0.86 and 0.88, respectively. Therefore, the reported results indicate that LMT performs better for bug severity prediction under the reported experimental setup. The superior performance of the LMT algorithm can be attributed to the fact that the number of selected features after applying feature selection method is relatively small compared to the original feature set. Therefore, discarding unimportant features leads LMT algorithm to build classification trees that are significantly smaller than the standard classification trees and has accurate prediction results.

Finally, the severity of new bug reports submitted to JIRA bug tracking system can be predicted automatically based on the reported prediction model. This automatic prediction would be beneficial for software engineers and developers of INTIX Company to automatically assign the severity of reported bugs instead of manual work. As a consequence, severe bugs will be solved on time without causing a delay for fixing them.

V. Conclusions

In this research paper, we described machine learning approaches for predicting the severity level of software bug reports in closed-source projects and leveraged Information Gain (IG) feature selection method to enhance the performance of the prediction by increasing the prediction accuracy of the bug severity level.

The procedure of the proposed methodology was illustrated and evaluated by comparing eight popular machine learning algorithms, namely Naive Bayes, Naive Bayes Multinomial, Support Vector Machine (SVM), Decision Tree (J48), Random Forest, Logistic Model Trees (LMT), Decision Rules (JRip) and KNN. The performances of utilized machine learning algorithms were evaluated in terms of accuracy, F-measure and Area Under the Curve (AUC). Furthermore, one of the objectives of this research work was to utilize feature selection techniques for the enhancement of selected features that will, in turn, allow for more precise discriminative power and produce better prediction performance results.

The proposed methodology was conducted on the existing severity levels assigned manually by the developers of INTIX Company through the JIRA bug tracking system. The experiments were evaluated using performance measures: accuracy, F-measure and AUC. The obtained results indicated that the Logistic Model Trees (DT: LMT) outperformed other classifiers and the overall performance has been enhanced after applying feature selection method. The results showed that the LMT algorithms reported the best performance results based on all performance measures (Accuracy= 86.31, AUC= 0.90, F-measure= 0.91).

In future work, we plan to adopt sentiment analysis and opinion mining techniques to enhance the performance of the process of predicting the severity level of software bug reports.

Acknowledgment

The authors are thankful to INTIX Company for providing us with the bug report dataset.

References

[1] T. Zhang, H. Jiang, X. Luo, and A. T. Chan, "A literature review of research in bug resolution: Tasks, challenges and future directions," The Computer Journal, vol. 59, pp. 741-773, 2016.

[2] C. Liu, J. Yang, L. Tan, and M. Hafiz, "R2Fix: Automatically generating bug fixes from bug reports," in 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, 2013, pp. 282-291.

[3] G. Murphy and D. Cubranic, "Automatic bug triage using text categorization," in Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering, 2004.

[4] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?," in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, 2008, pp. 308-318.

[5] Bugzilla. (2019). Bug Tracking System. Available: http://www.bugzilla.org/

[6] Atlassian. (2019). JIRA. Available: http://www.atlassian.com/software/jira/

[7] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in CASCON, 2008, pp. 304-318.

[8] J. Uddin, R. Ghazali, M. M. Deris, R. Naseem, and H. Shah, "A survey on bug prioritization," Artificial Intelligence Review, vol. 47, pp. 145-180, 2017.

[9] K. Chaturvedi and V. Singh, "Determining bug severity using machine learning techniques," in 2012 CSI Sixth International Conference on Software Engineering (CONSEG), 2012, pp. 1-6.

[10] 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.

[11] 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.

[12] I. Herraiz, D. M. German, J. M. Gonzalez-Barahona, and G. Robles, "Towards a simplification of the bug report form in eclipse," in Proceedings of the 2008 international working conference on Mining software repositories, 2008, pp. 145-148.

[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] 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.

[15] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in 2008 IEEE International Conference on Software Maintenance, 2008, pp. 346-355.

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

[17] A. F. Otoom, D. Al-Shdaifat, M. Hammad, and E. E. Abdallah, "Severity prediction of software bugs," in 2016 7th International Conference on Information and Communication Systems (ICICS), 2016, pp. 92-95.

[18] K. Jin, A. Dashbalbar, G. Yang, B. Lee, and J.-W. Lee, "Improving predictions about bug severity by utilizing bugs classified as normal," Contemp Eng Sci, vol. 9, pp. 933-942, 2016.

[19] C.-Z. Yang, K.-Y. Chen, W.-C. Kao, and C.-C. Yang, "Improving severity prediction on software bug reports using quality indicators," in 2014 IEEE 5th International Conference on Software Engineering and Service Science, 2014, pp. 216-219.

[20] V. Singh, S. Misra, and M. Sharma, "Bug severity assessment in cross project context and identifying training candidates," Journal of Information & Knowledge Management, vol. 16, p. 1750005, 2017.

[21] W. Liu, S. Wang, X. Chen, and H. Jiang, "Predicting the severity of bug reports based on feature selection," International Journal of Software Engineering and Knowledge Engineering, vol. 28, pp. 537-558, 2018.

[22] 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.

[23] G. Sharma, S. Sharma, and S. Gujral, "A novel way of assessing software bug severity using dictionary of critical terms," Procedia Computer Science, vol. 70, pp. 632-639, 2015.

[24] S. Sharmin, F. Aktar, A. A. Ali, M. A. H. Khan, and M. Shoyaib, "Bfsp: A feature selection method for bug severity classification," in 2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC), 2017, pp. 750-754.

[25] C.-Z. Yang, C.-C. Hou, W.-C. Kao, and X. Chen, "An empirical study on improving severity prediction of defect reports using feature selection," in 2012 19th Asia-Pacific Software Engineering Conference, 2012, pp. 240-249.

[26] 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.

[27] 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.

[28] 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.

[29] T. Liu, S. Liu, Z. Chen, and W.-Y. Ma, "An evaluation on feature selection for text clustering," in Proceedings of the 20th international conference on machine learning (ICML-03), 2003, pp. 488-495.

[30] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in Icml, 1997, p. 35.

[31] M. F. Caropreso, S. Matwin, and F. Sebastiani, "A learner-independent evaluation of the usefulness of statistical phrases for automated text categorization," Text databases and document management: Theory and practice, vol. 5478, pp. 78-102, 2001.

[32] S. M. Weiss, N. Indurkhya, T. Zhang, and F. Damerau, Text mining: predictive methods for analyzing unstructured information: Springer Science & Business Media, 2010.

[33] P. S. Laplace, "Memoir on the probability of the causes of events," Statistical Science, vol. 1, pp. 364-378, 1986.

[34] T. M. Mitchell, "Machine Learning," ed: Mcgraw-hill, 1997.

[35] A. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification," in AAAI-98 workshop on learning for text categorization, 1998, pp. 41-48.

[36] V. Christlein, D. Bernecker, F. Hönig, A. Maier, and E. Angelopoulou, "Writer identification using GMM supervectors and exemplar-SVMs," Pattern Recognition, vol. 63, pp. 258-267, 2017.

[37] S. Cogill and L. Wang, "Support vector machine model of developmental brain gene expression data for prioritization of Autism risk gene candidates," Bioinformatics, vol. 32, pp. 3611-3618, 2016.

[38] F. Simistira, V. Katsouros, and G. Carayannis, "Recognition of online handwritten mathematical formulas using probabilistic SVMs and stochastic context free grammars," Pattern Recognition Letters, vol. 53, pp. 85-92, 2015.

[39] D. N. Sotiropoulos, D. E. Pournarakis, and G. M. Giaglis, "SVM-based sentiment classification: a comparative study against state-of-the-art classifiers," International Journal of Computational Intelligence Studies, vol. 6, pp. 52-67, 2017.

[40] J. Platt, "Fast training of support vector machines using sequential minimal optimization, in, B. Scholkopf, C. Burges, A. Smola,(eds.): Advances in Kernel Methods-Support Vector Learning," ed: MIT Press, 1998.

[41] J. R. Quinlan, C4. 5: programs for machine learning: Elsevier, 2014.

[42] L. Breiman, "Random forests," Machine learning, vol.45,pp. 5-32, 2001.

[43] N. Landwehr, M. Hall, and E. Frank, "Logistic model trees," Machine learning, vol. 59, pp. 161-205, 2005.

[44] W. W. Cohen, "Fast effective rule induction," in Machine Learning Proceedings 1995, ed: Elsevier, 1995, pp. 115-123.

[45] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, Data Mining: Practical machine learning tools and techniques: Morgan Kaufmann, 2016.