

# Bug Severity Prediction using Keywords in Imbalanced Learning Environment

**Jayalath Ekanayake**

Dept. of Computer Science and Informatics, Faculty of Applied Sciences, Uva Wellassa University, Badulla, 90000, Sri Lanka  
E-mail: jayalath@uwu.ac.lk

Received: 22 October 2020; Revised: 26 December 2020; Accepted: 25 February 2021; Published: 08 June 2021

**Abstract:** Reported bugs of software systems are classified into different severity levels before fixing them. The number of bug reports may not be equally distributed according to the severity levels of bugs. However, most of the severity prediction models developed in the literature assumed that the underlying data distribution is evenly distributed, which may not correct at all instances and hence, the aim of this study is to develop bug classification models from unevenly distributed datasets and tested them accordingly.

To that end first, the topics or keywords of developer descriptions of bug reports are extracted using Rapid Keyword Extraction (RAKE) algorithm and then transferred them into numerical attributes, which combined with severity levels constructs datasets. These datasets are used to build classification models, Naïve Bayes, Logistic Regression, and Decision Tree Learner algorithms. The models' prediction quality is measured using Area Under Recursive Operative Characteristics Curves (AUC) as the models learnt from more skewed environments.

According to the results, the prediction quality of the Logistics Regression model is 0.65 AUC whereas the other two models recorded maximum 0.60 AUC. Though the datasets contain comparatively a smaller number of instances from the high severity classes; Blocking and High, the Logistic Regression models predict the two classes with a decent AUC value of 0.65 AUC. Hence, this project shows that the models can be trained from highly skewed datasets so that the models prediction quality is equally well over all the classes regardless of number of instances representing the class. Further, this project emphasizes that the models should be evaluated using the appropriate metrics when the models are trained from imbalance learning environments. Also, this work uncovers that the Logistic Regression model is also capable of classifying documents as Naïve Bayes, which is well known for this task.

**Index Terms:** Bug reports classification, bug severity level, topics modeling, candidate keywords, classification algorithms.

## I. Introduction

Typically, software systems released with defects as they become more and more complex. Usually, the users and the developers inform the software bugs to triggers through systems such as Bugzilla<sup>1</sup>, Jira<sup>2</sup>. Such bugs may be resolved in future revisions. Reporting bugs helps developer to fix the bugs in next release so that the users get an improved version of the software. However, the developer is under pressure if they receive too many bug reports from the users or other parties.

Usually, the triagers receive significant number of bug reports in daily basis [1]. However, due to many reasons the triagers require long time to fix the reports [2]. According to Wang et al. [3] the users reported average of 1000 bug reports for software projects in daily basis. Further Wang et al. reported that the developers spend two and half days per week to read 1000 bug reports. Reading all these reports and starts fixing them according to the order they received, without categorizing them into different severity levels, then some high severity bugs may reside in the software causing negative implications on the project. Hence, categorizing the bugs based on the severity level is essential; so that the high severity bugs are fixed immediately. A bug can be assigned to one of four severity levels: Blocking, High, Low, Normal as in UNIX Kernels project. The impact level of Blocking bugs is the highest whereas Normal is the lowest. The Blocking bugs are given the highest priority when fixing them, however, Normal bugs may remain on software projects for a long time. Typically, the bug reporters assign the severity level of the bug, or he may leave it blank. Perhaps, the reporter's decision may be incorrect as his judgment on the bug severity may differ from the triager's perception.

Manual bug classification process is not efficient, as bug triagers need to read the textual content of all bug reports

---

<sup>1</sup> <https://www.bugzilla.org>

<sup>2</sup> <https://www.atlassian.com>

and then compare the new reports with existing reports. In some bug reports the description is not sufficient to figure out the nature of the bug and hence, developers want to spend even more time to classify them. To that end, bug classification tools can be used to speed up the bug classification process and hence, high severity bugs can be fixed immediately. The automatic bug classification tools can be trained using machine learning algorithms. Typically, the severity level assigned in resolved bug reports is confirmed to be correct and hence, the data extracted from such reports can be used to train the machine learning algorithms for classifying new bug reports. The severity is divided into four levels and the number of bug reports assigned into different severity levels may not be evenly distributed. Hence, training the classification models from such an imbalanced dataset can be biased towards the class containing the majority of instances. Consequently, the models mostly predict the instances in the majority class with decent accuracy whereas the instances belong to the other classes may not be correctly classified. In such a skewed learning environment the testing sample may also contain the same distribution as the training sample. However, many of the existing bug severity prediction models trained from history data are neither mention the distribution of the datasets used to train the models, nor acknowledged this issue [7-14]. This implies that they assumed the underlying data distribution is balanced, which may not essentially be true in all real datasets such as I use in this project. However, reference [10] addressed this issue to a certain extent. They proposed a cost sensitive classification by defining two categories of classes; minority and majority based on the number of instances representing the two classes and the misclassification cost of minority class is higher than the majority class regardless of the significance of the classes. According to their definition the misclassification cost of the Blocking and High severity classes could be smaller than Normal and Low if the dataset contains more instances from Blocking and High classes. However, the bugs from these two classes are more important for the developers than the other two classes.

Hence, the objective of this project is to develop bug severity prediction models from imbalanced learning environments and test them properly using an appropriate metric. Many models developed in the literature [7-14] used the accuracy--correctly classified instances vs. the total number of instances presented--to evaluate the prediction quality in skewed learning environments [4,5]. However, the accuracy would not provide fair judgment on the quality of models trained from unevenly distributed datasets like used in this project. Alternatively, Area Under Recursive Operative Characteristics Curves (AUC) can be used to evaluate such prediction models, as the AUC does not depend on the underlying data distribution [6]. Further, the prediction quality is evaluated in category wise in contest to the overall prediction quality calculated in many prediction models in the literature.

This project proposes to construct three prediction models; Naïve Bayes, Logistic Regression, and Decision Tree Learner algorithms to categorize bugs into one of four severity levels--Blocking, High, Low, and Normal--. The three models predict the class probability for a given instance. Also, the models are trained from skewed datasets as shown in Table 1 and hence, the prediction quality is measured using AUC. The proposed models address the questionable point in most of the models developed in the literature.

## 2. Related Works

Software repository contains source code, bug reports, email archives etc. Mining these repositories is popular among research community in recent years as it extracts hidden patterns, which may be non-trivial, previously not known and vital information for the developers.

There are models that predict bug severity using textual description in bug reports [7-14].

Reference [7] carried out a comparison between the newly proposed bug and already existing bugs in the bug database using BM25-based document similarity function and then assigned the severity for the new reports based on the similarity calculated. The performances of the similarity calculation models were measured only using precision, recall and F-measure while acknowledging this fact as a limitation of this study. However, the initial distribution of the severity levels was not considered when calculating the prediction quality of models.

Roy and Rossi [8] proposed a feature selection method, which enhances the prediction accuracy of severity prediction models. Features were developed based on textual components along with bi-grams and trained the Naïve Bayes (NB) classifier using Mozilla and Eclipse data and revealed that there was an improvement in prediction quality due to the feature selection method. Further, they showed that the accuracy may vary on projects and adding further bigrams could degrade the performances.

Reference [9] compiled improved versions of bug reports of Mozilla, Eclipse and GCC by integrating questions and the corresponding answers from stack trace. Subsequently, they trained Naïve Bayesian, k-Nearest Neighbor algorithm (KNN), and Long Short-Term Memory (LSTM) models to classify bug reports into severity levels. The performances of the models were evaluated using f-measure and concluded that average F-measure of Mozilla, Eclipse, and GCC projects is improved by 23.03%, 21.86%, and 20.59% respectively compared to the previous studies. Further, the experiment showed that the Naïve Bayesian based approach outperformed all baseline approaches. Tan et al. assumed that the bug report distribution in terms of severity levels was equal and hence, did not influence on prediction quality, which may not be correct all the time.

Reference [10] amalgamated the stack trace information of the bug reports together with categorical features such as component, OS, product to enhance the quality of bug reports. They labeled the classes as under-sampled and over-

sampled based on the number of instances representing each class and then assigned high misclassification cost for under-sampled classes and low misclassification cost for over-sampled classes in order to address the class imbalance distribution problem. The misclassification cost was the ratio between the number of instances of the majority class and the number of instances of the class considered. They used K Nearest Neighbor (KNN) and obtained the probability of classifying an unknown instance into given number of classes. Finally, the cost of assigning an instance into the given class was calculated by multiplying the misclassification cost and the probability of assigning an instance assigned into a class by KNN. The class label with lowest misclassification cost was selected as the class label for the unknown instance. The accuracy of the model was measured using precision, recall and F-measure as usual. According to the results the models improved the accuracy (F-measure) up to 35% when integrating the stack trace information into the category features of bug reports. Compare to the above study I address the class imbalanced problem in a different way. Further, I use only stack trace information to classify the bug reports whereas ref. [10] used both stack trace and categorical features of the bug reports.

Reference [11] proposed a method to assign severity levels for bugs in early stage of the development project. They considered the way the bug report was written rather than the content of the bug report to predict the severity level of new bugs. The outcome of the project proved that the method was able to predict the severity of bugs in early stage of the development and also able to outperform an existing keyword-based classifier used in many NASA projects.

Reference [12] used Shannon entropy measure to calculate uncertainty of bug summary. They trained classifiers using bug priority, along with summary weight and entropy to predict severity of bugs of new projects, which do not have history records. The performance of the classifiers was measured using accuracy and f-measure. The method showed improved prediction quality of accuracy and F-measure 70% and 93%. However, the initial distribution of the training dataset was not taken into consideration when selecting the performance metrics.

Reference [13] proposed a method using Bellwether analysis to extract terms from the bug summary and trained four models: deep neural network, logistic regression, k-nearest neighbor, and random forest. The models' performances were evaluated using the standard metrics such as precision, recall and F-measure. The result showed the model achieved the F-measure ranging from 14% to 97%. Also, reference [14] proposed a new method to assign severity levels for bugs using swarm intelligence and machine learning. The optimum feature set was extracted from bug summary and trained Naive Bayes models to classify bug reports. The performances were analyzed using three metrics precision, recall and F-measure as in reference [13] and concluded that the bug reports having more content in summarization process increase the severity classification accuracy.

Similar to bug severity prediction, bug priority prediction is also an important task. The user defines the bug severity whereas the developer defines priority. Many models have been proposed in literature for bug priority prediction [15-22]. These models also shared a common procedure as severity prediction models except reference [22]. Also, several models have been developed for bug prediction in software systems [28-30].

Further, many models have been developed for document processing [26,27]. However, this project uses the method as described in reference [23].

The above referenced models often used metrics; precision, recall and F-measure to denote the performances of the models without providing sufficient information about the underlying data distribution except references [10,22]. Also, many of the models provided overall prediction quality instead of class level. Hence the performances of the prediction models especially on minority classes may not be clearly represented in the overall prediction quality when in skewed learning environments. Further, it is always a good practice to take output probability values (if the models output class probabilities for a given instance) into consideration when measuring the performance of prediction models. The ROC curve penalizes the wrong predictions with high probability values by reducing the score for Area Under Curve (AUC). Though a model gains high accuracy, the AUC may be low as the model makes wrong predictions with high probability values. But the metrics; precision, recall and f-measure may not be sensitive for the output probabilities as ROC curves.

Summarizing, the common procedure followed by most of the above models is the models were trained using history data and tested using another dataset without much consideration on the underlying data distribution. However, this project evaluates the models considering the underlying data distribution. Hence, this project uses Area Under ROC curves to measure the performances of the prediction models.

### 3. Recursive Operative Characteristics Curves (ROC)

This project uses Area Under Curve (AUC) of ROC to evaluate the models as they are trained using highly skewed dataset (See Table 1). Fig. 1 shows an example ROC curve. The Y-axis of the ROC curve indicates the true positives (Sensitivity), which the instances are from positive group and the model also predicts as positive, contained in the test sample as a fraction (percentage) of the total number of positives. Whereas the X-axis of the ROC curve plots the number of negatives (Specificity), which the instances are from positive, but the model predicts as negative, as a fraction (percentage) of the total number of negatives. The ROC curve of a perfect model passes through (0,1) corner. Hence, the ROC curve closer to upper left corner provides that the prediction quality of the model is perfect. Further, it shows that the model predicts the class for a given instance with a high probability is correct, which is a good characteristic of a model. The prediction along with diagonal, which the AUC around 0.5 or below, indicates random

prediction whereas AUC=1.0 is perfect.

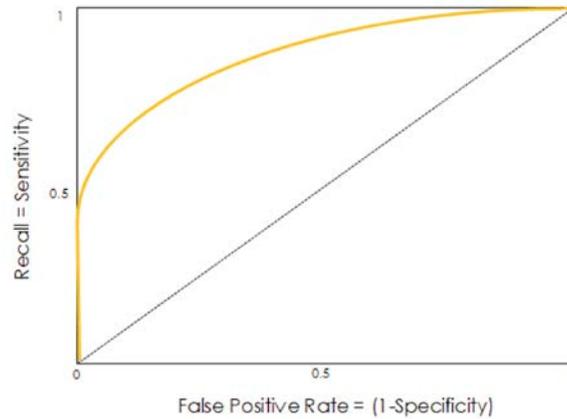


Fig.1. ROC Curve (Source: <https://miro.medium.com>)

#### 4. Bug Reports

The aim of this project is to train bug severity prediction models from imbalanced training datasets. The training datasets are created using the bug reports. The severity level is predicted using the features extracted from the developer description. Hence, the dependent variable is the severity level whereas the developer description is the predictor or independent variables. Further, the prediction models are needed to train from skewed datasets. The UNIX Kernels project<sup>3</sup> contains more than 16000 bug reports and the distribution of bug reports is uneven. Also, the UNIX Kernel project has more than 20 years of history and it is essential for this kind of study.

The bug reports are implemented in XML format and stored them in a database. A bug report has a fixed format containing many categorized fields along with a summary statement of the bug and the description or the comments made by the developers. The bug ID is the unique identification of a bug report. A bug typically appears in a component, which is subset of a product. A reported bug is initially assigned to a developer and the developer's name is mentioned in the assignee field. The bug status indicates whether the bug is resolved, verified or closed. The operating system, which the software crashed, resolution, reporter of the bug, hardware, changed data, version, priority and severity are also recorded in the bug report.

#### 5. Classification Algorithms

This project trained three classification algorithms; Naïve Bayes, decision tree learner, and logistic regression to classify bug reports into severity levels. The models assign a probability for a given test instance being classified into a class.

**Naïve Bayes:** This model is a derivation of Bayer's rule of conditional probability as in (1).

$$P(C | x) = \prod_{i=1}^n P(x_i | C)P(C) \quad (1)$$

Where  $X = (x_1, x_2, \dots, x_n)$  are the evidence and  $C$  is the class. The Naïve Bayes assumes that the attributes  $x_1, x_2, \dots, x_n$  are independent of each other. However, this assumption cannot be achieved in reality. According to the equation (1) the class probability of a given instance  $x$  is calculated based on the previous evidence and then the instance is assigned to the class with highest probability. This project selected Naïve Bayes model as it has shown decent performances in document classification.

**Decision Tree:** This model creates a tree like structure so that the attributes of the dataset are mapped into the nodes of the tree. The root node, which is the first node of the tree, divides the dataset at first level. This process continues until the leaves of the tree contain all the similar type of instances or the majority of instances is in one type. The decision trees are human readable and hence, are heavily applied in classification and pattern recognition fields.

<sup>3</sup> <https://bugzilla.kernel.org>

**Logistic Regression:** This model is a modification of the multi response linear regression. However, the Logistic Regression model transforms the target variable in between 0 and 1 as shown in equation 2.

$$\Pr[1|a_1, a_2, \dots, a_k] = \frac{1}{(1 + \exp^{(-w_0 - w_1 a_1 - w_k a_k)})} \quad (2)$$

Where  $w_0, \dots, w_k$  are weights, which are determined from the training data and  $a_1, \dots, a_k$  are the attributes. The weights are determined using log-likelihood contrary to the regression models that use squared-error. The 0.5 probability value is used as the decision boundary in two-class classification problem.

The common feature of the above models is that the models calculate output a probability of given instance being classified into a class. Further, these models are often used in classification problems.

## 6. Methodology

### 6.1. Dataset Preparation

The datasets are prepared using the bug reports in the UNIX Kernels project. The bug reports are downloaded using a java XML passer. Total 16400 bug reports are downloaded but only 10407 bug reports are considered, as the other bug reports are invalid or not yet been verified. Severity level distribution of bug reports is shown in Table 1. Accordingly, the distribution is highly biased towards the severity level normal as many bugs have been classified into normal. The developer description contains keywords, which are extracted using the RAKE algorithm [23]. The stop words and delimiters are used to tokenize the textual description. A list of contiguous words is prepared through categorizing the tokens, which are created from the previous step. This list of contiguous words is known as candidate keywords. The Rapid Keyword Extraction algorithm identifies the candidate keywords containing interior stop words such as *axis of evil*. However, such words should appear more than one time in the description. Long candidate keywords are more useful for classifying documents; hence this project uses long candidate keywords. The candidate keywords are ranked as the high rankers are taken into the feature list. The ranking can be done according to the reference [23]. Accordingly, the ratio of degree to frequency is used to rank the candidate keywords as the RAKE is biased to extract long candidate keywords. The final set of candidate keywords is selected based on the scores obtained by the keywords. As proposed by Mihalcea and Tarau [24] the top one third of the candidate keywords is selected as the final set of keywords. The attribute set for the prediction algorithms is determined from the first batch of candidate keywords. The candidate keywords are converted into a set of numerical features using the string-to-word filter implemented in WEKA [25] and integrated the severity level of the bug reports to prepare a label dataset typically called the training dataset.

Table 1. Distribution of severity labels

Normal	Blocking	High	Low
7921	551	1655	509

### 6.2. Training and Testing Models

The classification models implemented in WEKA [23], are trained and tested using stratified 10-fold cross validation that preserves the initial distribution of the datasets. The prediction performance of the models is measured using AUC in category basis as opposed to overall quality. First, the models are trained using 5 attributes and gradually incremented the number by 5.

## 7. Results and Discussion

The objective of this project is to develop bug severity prediction models from imbalance learning environments. The provided datasets are highly skewed so that the datasets are representing significantly a smaller number of bug reports from the two most important severity categories; Blocking and High. Hence, the prediction quality of the models should be evaluated with an appropriate metrics. To that end I use Area Under Curve (AUC) of Receiver Operative Characteristics (ROC) as it does not sensitive for the underlying data distribution.

Bug reports are classified into four deferent levels--normal, blocking, low, high--. The classification task is conducted using three models: Naïve Bayes, decision tree learner, and logistic regression. The models are trained from the datasets created using bug reports of UNIX Kernels project. The models classify a given instance based on the calculated class probability.

The datasets can contain any number of attributes. However, training a prediction model using a large number of attributes may reduce its performances. Therefore, the prediction algorithms are initially trained using five attributes and subsequently, the number of attributes presented to train the models is incremented by 5. Tables 2, 3 and 4 show the

classification accuracies of the models measured in AUC.

Table 2. Prediction quality variability of the Naïve Bayes models corresponding to the number of attributes

Model	# of Attributes	Prediction Quality (AUC)			
		Normal	Blocking	Low	High
Naïve Bayes Model	05	0.52	0.54	0.59	0.57
	10	0.48	0.60	0.58	0.59
	15	0.48	0.60	0.56	0.60
	20	0.48	0.60	0.57	0.59
	25	0.47	0.59	0.57	0.59
	30	0.47	0.59	0.55	0.60
	35	0.47	0.60	0.55	0.59
	40	0.47	0.60	0.55	0.60
	45	0.47	0.60	0.55	0.60

Table 3. Prediction quality variability of decision tree models corresponding to the number of attributes

Model	# of Attributes	Prediction Quality (AUC)			
		Normal	Blocking	Low	High
Decision Tree Learner	05	0.50	0.50	0.50	0.50
	10	0.55	0.57	0.54	0.55
	15	0.55	0.53	0.54	0.55
	20	0.54	0.53	0.54	0.53
	25	0.54	0.54	0.56	0.54
	30	0.53	0.52	0.55	0.54
	35	0.55	0.53	0.60	0.52
	40	0.54	0.51	0.60	0.52
	45	0.54	0.51	0.59	0.52

Table 4. Prediction quality variability of logistic regression models corresponding to the number of attributes

Model	# of Attributes	Prediction Quality (AUC)			
		Normal	Blocking	Low	High
Logistic Regression	05	0.56	0.56	0.59	0.59
	10	0.60	0.63	0.59	0.63
	15	0.60	0.64	0.60	0.64
	20	0.61	0.64	0.59	0.64
	25	0.61	0.64	0.60	0.64
	30	0.62	0.64	0.60	0.64
	35	0.62	0.65	0.61	0.64
	40	0.62	0.65	0.61	0.64
	45	0.62	0.65	0.62	0.64

According to the results the logistic regression model outperforms the other two models: Naïve Bayes and Decision tree learner providing decent classification accuracy around 0.65 AUC. This model achieves the maximum AUC using 35 attributes and adding further attributes does not show any impact in the prediction quality.

Blocking and High severity bugs have a negative impact on software projects and hence, such bugs should be fixed as soon they found. Also, prediction of such bugs is very helpful for developers as they can allocate sufficient resources and time for fixing them. The Logistics Regression model predicts the Blocking and High severity classes more accurately than the other two classes (see Table 4) though the training sets contain significantly smaller number of instances representing these two classes, which is a notable finding in this research.

The prediction quality of the Naïve Bayes model is slightly above random classifier. Also, the model provides maximum AUC with five attributes. Unlike other two models increasing the number of attributes negatively impacts on the prediction quality of Naïve Bayes models. The decision tree algorithm classifies bug reports with better AUC values compared to the Naïve Bayes model. Further, the algorithm reaches maximum AUC using 10 attributes and presenting more attributes to the algorithm does not enhance the prediction quality.

Reference [10] addressed the imbalance class label problem using a cost sensitive classification method, which a wrong prediction in a minority class is more costly than a wrong prediction in a majority class. The minority class contains a smaller number of instances whereas the majority class represents a larger number of instances. Accordingly, the categorization is purely based on the number of instances whereas this project considers each class is equally

important. Our evaluation method highly penalizes a wrong classification by the models with a high probability value regardless of the class, which is a fair rule. Though this project considers predicting all the classes are equally important, the High and Blocker severity bugs are very crucial for the developers, hence, predicting them accurately is very helpful for the developers. The Logistics Regression models developed in this project predicts these two classes more accurately than the other classes. This is a notable feature of our project, and this was not discussed in the reference [10].

It is a well-known fact that the Naïve Bayes algorithm is frequently applied for document classification. However, the results of this project reveal that the logistic regression algorithm is also a suitable for document classification. Though the models are trained and tested in highly imbalanced learning environment, they perform reasonably well and hence, the logistic regression model also can be used for the bug report classification task.

This project reveals that the class imbalance problem can be addressed using a proper evaluation metrics rather considering the number of instances representing in the datasets.

## 8. Conclusion

The aim of this project to develop bug severity classification models in imbalance learning environments. The bug reports are extracted from the Unix Kernels project. The classification is conducted using the developer description of bug reports. The developer description contains keywords, which are process by the RAKE algorithm and then transferred them into numerical attributes using a string-to-word filter implemented in WEKA [23]. These attributes are used to train three classification models: Naïve Bayes, decision tree learner and Logistics Regression models. As the models are trained on highly imbalanced datasets the prediction quality of models should be measured using an appropriate metrics. Accordingly, the AUC is used to measure the prediction quality of models. The results show that the logistic regression model predicts the two severity classes: Blocking and High more accurately than the other two models with a decent AUC value. Typically, the Naïve Bayes algorithm is well known for document classification. But this project uncovers that the logistic regression is also a reliable approach for document classifications.

This project highlighted the need of evaluating the models developed in highly imbalanced learning environment using a proper matrix. Further, this project showed that the accuracy or f-measure would not provide a fair evaluation of models when the learning environment is highly skewed. Moreover, if the models calculate the prediction probability, then the misclassification cost can be quantified by taking the probability value into consideration and penalized the model, which is a fair justice for the models. These findings can be applied for future prediction models in any domain trained from either balance or imbalance distribution.

The misclassification cost of different classes can be varied. As an example, the misclassification cost of Blocking and High severity classes is higher than the misclassification cost of Normal and Low severity classes. However, when evaluating the prediction quality this project did not consider the misclassification cost of each class. This can be addressed in future projects by first defining the unit misclassification cost of each class and later combined it with the prediction probability.

## References

- [1] Xie T, Zhang L, Xiao X, Xiong YF, Hao D. Cooperative software testing and analysis: Advances and challenges. *Journal of Computer Science and Technology*. 2014 Jul 1;29(4):713-23.
- [2] Xia X, Lo D, Wen M, Shihab E, Zhou B. An empirical study of bug report field reassignment. In2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE) 2014 Feb 3 (pp. 174-183). IEEE.
- [3] Wang J, Wang S, Cui Q, Wang Q. Local-based active classification of test report to assist crowdsourced testing. InProceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering 2016 Aug 25 (pp. 190-201).
- [4] Davis J, Goadrich M. The relationship between Precision-Recall and ROC curves. InProceedings of the 23rd international conference on Machine learning 2006 Jun 25 (pp. 233-240).
- [5] Lessmann S, Baesens B, Mues C, Pietsch S. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*. 2008 May 23;34(4):485-96.
- [6] Provost F, Fawcett T. Robust classification for imprecise environments. *Machine learning*. 2001 Mar 1;42(3):203-31.
- [7] Tian Y, Lo D, Sun C. Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In2012 19th Working Conference on Reverse Engineering 2012 Oct 15 (pp. 215-224). IEEE.
- [8] Roy NK, Rossi B. Towards an improvement of bug severity classification. In2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications 2014 Aug 27 (pp. 269-276). IEEE.
- [9] Tan Y, Xu S, Wang Z, Zhang T, Xu Z, Luo X. Bug severity prediction using question-and-answer pairs from Stack Overflow. *Journal of Systems and Software*. 2020 Mar 2:110567.
- [10] Sabor KK, Hamdaq M, Hamou-Lhadj A. Automatic prediction of the severity of bugs using stack traces and categorical features. *Information and Software Technology*. 2020 Jul 1; 123:106205.
- [11] Arokiam J, Bradbury JS. Automatically predicting bug severity early in the development process. InProceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results 2020 Jun 27 (pp. 17-20).
- [12] Kumari M, Singh UK, Sharma M. Entropy Based Machine Learning Models for Software Bug Severity Assessment in Cross Project Context. InInternational Conference on Computational Science and Its Applications 2020 Jul 1 (pp. 939-953). Springer, Cham.

- [13] Kudjo PK, Chen J, Mensah S, Amankwah R, Kudjo C. The effect of Bellwether analysis on software vulnerability severity prediction models. *Software Quality Journal*. 2020 Jan 7:1-34.
- [14] Kukkar A, Mohana R, Kumar Y. Does bug report summarization help in enhancing the accuracy of bug severity classification?. *Procedia Computer Science*. 2020 Jan 1; 167:1345-53.
- [15] Kanwal J, Maqbool O. Bug prioritization to facilitate bug report triage. *Journal of Computer Science and Technology*. 2012 Mar 1;27(2):397-412.
- [16] Alenezi M, Banitaan S. Bug reports prioritization: Which features and classifier to use?. In 2013 12th International Conference on Machine Learning and Applications 2013 Dec 4 (Vol. 2, pp. 112-116). IEEE.
- [17] Tian Y, Lo D, Xia X, Sun C. Automated prediction of bug report priority using multi-factor analysis. *Empirical Software Engineering*. 2015 Oct 1;20(5):1354-83.
- [18] Kumari M, Singh VB. An improved classifier based on entropy and deep learning for bug priority prediction. In *International Conference on Intelligent Systems Design and Applications 2018 Dec 6* (pp. 571-580). Springer, Cham.
- [19] Waqar A. Software Bug Prioritization in Beta Testing Using Machine Learning Techniques. *Journal of Computers for Society* 2020;1(1):24-34.
- [20] Cheng X, Liu N, Guo L, Xu Z, Zhang T. Blocking Bug Prediction Based on XGBoost with Enhanced Features. In 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC) 2020 Jul 13 (pp. 902-911). IEEE.
- [21] Sharma M, Kumari M, Singh VB. Bug Priority Assessment in Cross-Project Context Using Entropy-Based Measure. In *Advances in Machine Learning and Computational Intelligence 2020* (pp. 113-128). Springer, Singapore.
- [22] Ekanayake, J.B., 2021. Predicting Bug Priority Using Topic Modelling in Imbalanced Learning Environments. *International Journal of Systems and Service-Oriented Engineering (IJSSOE)*, 11(1), pp.31-42.
- [23] Rose S, Engel D, Cramer N, Cowley W. Automatic keyword extraction from individual documents. *Text mining: applications and theory*. 2010 Mar 26; 1:1-20.
- [24] Mihalcea R, Tarau P. TextRank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing 2004 Jul* (pp. 404-411).
- [25] Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*. 2009 Nov 16;11(1):10-8.
- [26] Mai Farag Imam, Amal Elsayed Aboutabl, Ensaf H. Mohamed, "Automating Text Simplification Using Pictographs for People with Language Deficits", *International Journal of Information Technology and Computer Science*, Vol.11, No.7, pp.26-34, 2019.
- [27] Pierre MOUKELI MBINDZOUKOU, Arsène Roland MOUKOUKOU, David NACCACHE, Nino TSKHOVREBASHVILI, "A Stochastic Model for Simple Document Processing", *International Journal of Information Technology and Computer Science*, Vol.11, No.7, pp.43-53, 2019.
- [28] Ahmed Iqbal, Shabib Aftab, "Prediction of Defect Prone Software Modules using MLP based Ensemble Techniques", *International Journal of Information Technology and Computer Science*, Vol.12, No.3, pp.26-31, 2020.
- [29] Ekanayake J, Tappolet J, Gall HC, Bernstein A. Time variance and defect prediction in software projects. *Empirical Software Engineering*. 2012 Aug;17(4):348-89.
- [30] Ekanayake J, Tappolet J, Gall HC, Bernstein A. Tracking concept drift of software projects using defect prediction quality. In 2009 6th IEEE International Working Conference on Mining Software Repositories 2009 May 16 (pp. 51-60). IEEE.

## Authors' Profiles



**Jayalath Ekanayake** earned his BSc and MSc in Sri Lanka and his PhD in computer science at the University of Zurich, Switzerland.

He is a lecturer in Computer Science at the Uva Wellassa University, Sri Lanka. His research interest is pattern recognition.

Dr. Ekanayake is an IEEE member and also a life member of Sri Lanka Association of Advancement of Science (SLAAS).

**How to cite this paper:** Jayalath Ekanayake, "Bug Severity Prediction using Keywords in Imbalanced Learning Environment", *International Journal of Information Technology and Computer Science (IJITCS)*, Vol.13, No.3, pp.53-60, 2021. DOI: 10.5815/ijitcs.2021.03.04