Bug Report Severity Level Prediction in Free Libre/Open Source Software: A Survey and New Perspectives

Author1 Affiliation1 Author2 Affiliation2 Author2 Affiliation2

Abstract

Software evolution and maintenance activities in today Free/Libre Open Source Software rely primarily on information extracted from reports registered in bug tracking systems. The severity level attribute of a bug report is considered one of the most critical variables for planning these activities. For example, this attribute measures the impact the bug has on the successful execution of the software system and how soon a bug needs to be addressed by the development team. An incorrect assignment of severity level may generate a significant impact on the maintenance process. However, methods to assign severity level remains essentially manual with a high degree of subjectivity. Because it depends on the experience and expertise of whom have reported or reviewed the bug report, it may be considered a quite error-prone process. Due to its evident importance, both business and academic community have demonstrated a great interest in this topic, yielding an extensive research effort to provide methods to predict the bug report severity automatically. This paper aims to provide a comprehensive review of recent research efforts on automatically bug report severity prediction. To the best of our knowledge, it is the first review to categorizes quantitatively more than ten aspects of the experiments reported in several papers on bug report severity prediction. The gathered data confirm the relevance of this topic, reflects the scientific maturity of the research area, as well as, identifies gaps, which can motivate new research initiatives. ~~Besides~~, the results presented in this review have demonstrated that most of the surveyed papers proposed methods to predict severity level which extract features from unstructured text information. At the same time, they showed that traditional machine learning algorithms and text mining methods had played a central role in investigated experiments. However, ~~the scenario presented show~~ there is room ~~to obtain better~~ results using ~~more advanced~~ machine learning and text mining algorithms and techniques.

Index Terms

software maintenance; bug tracking systems; bug reports; severity level prediction; software repositories.

I. Introduction

Bug Tracking Systems (BTS) have been playing a key role as a communication and collaboration tool in Closed Source Software (CSS) and Free/Libre Open Source Software (FLOSS). In both development environments, the planning of software evolution and maintenance activities rely primarily on information of bug reports registered in this kind of system. It is particularly true in FLOSS, which is characterized by the existence of many users and developers with different levels of expertise spread out around the world, who might create or be responsible for dealing with several bug reports [1].

A user interacts with a BTS often through a simple mechanism called bug report form, which enables to communicate a bug to those in charge of development or are responsible for maintaining the software [2]. Initially, he or she should inform a short description, a long description, and an associated severity level (e.g., blocker, critical, major, minor, and trivial). Subsequently, a software team member will review this bug report and confirm or refuse it for some reason (e.g., bug report duplication). If the bug report is confirmed, the team member should provide more information to complement the bug report form, for example, by indicating its priority and by assigning a person who will be responsible for fixing the bug.

The information of severity level is recognized as a critical variable to estimate a prioritization of bug reports [3]. It measures the impact the bug has on the successful execution of the software system and defines how soon the bug needs to be addressed [4]. However, the severity level assignment remains mostly a manual process, which relies only on the experience and expertise of the person who has opened the bug report [1], [3], [4].

Moreover, the number of bug reports in large and medium software FLOSS projects is frequently very large [5]: The Eclipse project, for example, received over 2,764 requests and the GNOME project received over 3,263 requests between 01/10/2009 and 01/01/2010. Therefore, a manual assignment of severity level may be considered a quite cumbersome and error-prone process, and a wrong decision throughout bug report lifecycle may strongly disturb the planning of maintenance activities. For example, an important maintenance team resource could be allocated to address less significant bug reports before the most important ones.

Due to the evident importance of bug report severity information for the planning of FLOSS maintenance, both business and academic community have demonstrated great interest, and plenty of research has been done in this field. However, there are few mapping reviews well shaping this area and positioning existing works and current progress. Although these existing works helped understanding the challenges and opportunities in this research area, they suffer from one shortcoming: existing reviews lack in-depth coverage of the different dimensions of bug report severity prediction. The current systematic mapping

review fills this gap, by investigating and analyzing relevant papers — published from 2010 to 2017 — related to bug report severity prediction. To the best of our knowledge, it is the first review to provide a detailed document with the analysis of more than ten relevant aspects of the surveyed experiments, including: granularity of output class, FLOSS repositories, features and feature selection methods, text mining methods, machine learning algorithms, performance evaluation measures, sampling techniques, statistical tests, experiment tools, and type of solution.

A systematic mapping review aims to characterize the state-of-the-art on the prediction of bug report severity. This kind of paper provides a broad overview of a research area to determine whether there is research evidence on a particular topic. According to Kitchenham et al. [6] and Petersen et al. [7], results yielded by a mapping review help recognizing gaps to suggest future research and provide a direction to engage in new research activities appropriately. The key contributions of the current review are three-fold:

- It proposes a categorization scheme to position and organizes the experiments reported in papers on bug report severity prediction. For example, it provides a taxonomy to categorize the severity prediction problem type based on the granularity severity level which one wishes to predict.
- It provides an overview of the state-of-the-art research, summarizing the patterns, procedures, and methods employed by researches to predict bug report severity. For example, it indicates which are the most used machine learning algorithms in reported experiments.
- It discusses the challenges and open issues and future directions in this research field to share the visions and expand the horizons of bug report severity prediction.

The remaining of this paper is organized as follows: Section II provides the basic information background necessary to understand the research area. Section III presents related works that are relevant to our research. Section IV describes our research method. Section V presents our results. Section VI presents final findings and discussion. Finally, Section VII concludes the paper.

II. TERMINOLOGY AND BACKGROUND CONCEPTS

This section comments on general concepts necessary to understand this research area. More specifics concepts will be explained as they are cited in the document.

A. Bug Tracking System (BTS)

BTS [4] is a software employed to keep the record and track information about change requests, bug fixes, and support that could occur during the software life cycle. Usually, while reporting a bug, a user is asked to provide information about the bug by filling out a form, typically called bug report form.

1) Bug Report: Although there is no agreement on the terminology or the amount of information that users must provide to fill a bug report, they often describe their needs in popular BTS (e.g., Bugzilla, Jira, and Redmine) [3] filling at least the following attributes shown in Table I.

TABLE I: Commom attributes in a bug report form.

Type	Type of report (e.g., bug, improvement, and new feature)
Summary	Short description of report in one line.
Description	Long and detailed description of report in many lines of text. It could include source code snippets and stack tracing reports.
Severity	Level of severity of report (e.g., blocker, critical, major, minor and trivial).

After the user has registered the bug report, a development team will assess it. They may confirm or, for some reason (e.g., duplication), do not confirm this bug. In case of approving, they should complement the information of report with, for example, the assignment of a person as responsible for handling this request or a new severity level for the report. Typically, the sequence of steps a bug report goes through is modeled as a state machine. Figure 1a shows an example of such state machine, with a typical set of states a bug report can visit during its lifecycle. Each transition in the diagram triggers because of a specific event such as, for instance, "the bug has been resolved". All changes occurred in a bug report are stored in a repository, keeping valuable historical information about a particular software.

Both a user and a development team member can assign or reassign the severity level for a bug report during the lifecycle. Figure 1b illustrates how they should choose a correct severity level for a bug report based on Bugzilla guideline. To predict severity level, researchers sometimes aggregate these levels in severe (blocker, critical, and major) and non-severe (normal, minor and trivial) to work with a coarse-grained classification problem. Furthermore, some of them ignore the default severity level (often "normal") because they consider this level as a choice made by users when they have not sure about correct severity level. At other moments, researchers may attempt to predict a bug report severity level as blocking or non-blocking bug. A blocking bug is one that prevents others bugs to be fixed [9].

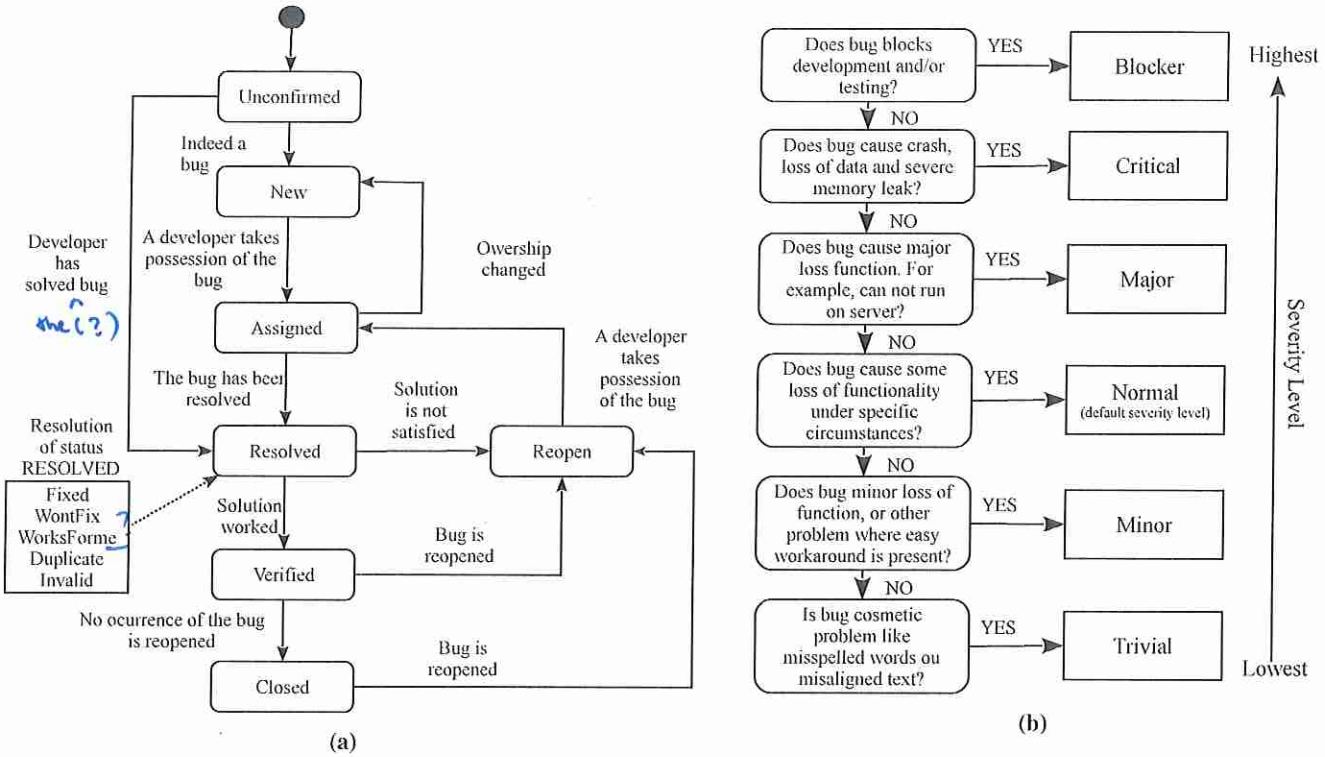

Fig. 1: (a) The life cycle of a bug report [8]. (b) Bugzilla guideline for bug report severity level assignment (https://www.bugzilla.org, As of July 3, 2018).

B. Machine Learning

Machine Learning (ML) [10] is an application of Artificial Intelligence (AI) that provides systems the ability to learn and improve from experience without being explicitly programmed. There are two types of ML algorithms: predictive (or supervised) and descriptive (or unsupervised). A predictive algorithm constructs a model based on historical data and uses this model to predict, from the values of input attributes, an output label (class attribute) for a new sample. A predictive task is called classification when the label value is discrete, or regression when the label value is continuous. On the other hand, a descriptive algorithm explores or describes a dataset. There is not an output label associated with a sample. Data clustering and pattern discovery are two examples of descriptive tasks. Bug report severity prediction is considered a classification problem; therefore, more detailing of descriptive algorithms are outside the scope of this paper.

1) ML algorithms: A ML algorithm works over a dataset which contains many samples or instances x_i, where $i = \{1..n\}$. Each instance is composed of $\{x_{i1}, x_{i2}, ..., x_{id}\}$ input attributes or independent variables, where $d = \{1..m\}$, and one output attribute or dependent variable, $x_{i(m+1)}$. Input attributes are commonly named features and output attribute as commonly named class. The most traditional ML classification algorithms are k-Nearest Neighbors, Näive Bayes, Decision Tree, Neural Networks, Random Forest and Support Vector Machine. In practice, they can be applied for both classification and regression tasks. However, this mapping review regards them only in the classification scenario. Next, a brief description of each one is presented [11]:

- **k-Nearest Neighbors** (k-NN) process the available instances (or neighbors) in a dataset and classifies a new instance based on its similarity measure to the k-nearest neighbors. Usually, k-NN algorithms utilize an Euclidean distance to quantify the proximity of neighbors. To calculate this distance, each input attribute of each instance in a dataset should represent points of an n-dimensional space.
- **Näive Bayes** (NB) decides which class an instance belongs to based on Bayesian Theorem conditional probability. The probabilities of an instance belonging to each of the C_k classes given the instance x is $P(C_k|x)$. Näive Bayes classifiers assume that given the class variable, the value of a particular feature is independent of the value of any other feature.
- **Decision Tree** consists of a collection of internal nodes and leaf nodes, organized in a hierarchical model. Each internal node corresponds to a feature, and each leaf node corresponds to a class label. The decision tree classifiers organized a series of test questions and conditions in a tree structure. A decision tree represents the model capable of guiding the decision making on the determination of the class to which an individual belongs.
- **Neural Network** is a learning algorithm that is inspired by the structure and functional aspects of biological neural networks [12]. It structures its processing as an interconnected group of artificial neurons. Figure 2 presents an artificial neuron model that contains a set of weighted inputs (w_i) that corresponds to the synapses; an adder function ($\sum$) that

sums the inputs signals; and an activation function ($f(.)$) that decides whether the neuron fires for the current inputs to generate the output value (y_o). Independent features of each instance represent neuron dendrites.

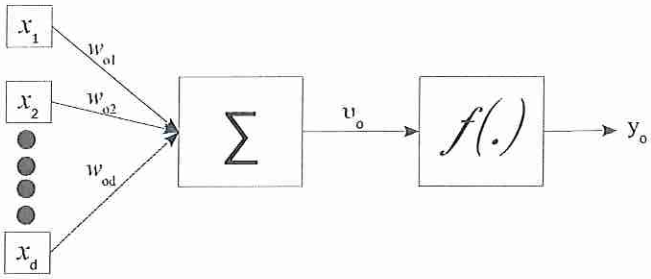

Fig. 2: A artificial neuron model based on Marsland [11].

- **Support Vector Machine(SVM)** represents each instance as points in an n-dimensional space. The algorithm divides every point of the separate categories by a gap that is as wide as possible [11]. This space between these two regions delimits the margin between classes. In Figure 3, for example, SVM will search for support vectors, which are instances located at the edge of an area in space. Such vectors define boundaries between one of these class instances (e.g., the squares) and another class ~~instances~~ (e.g., the circles). Each region contains instances ~~with the same value to its~~ class [13]. New instances are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

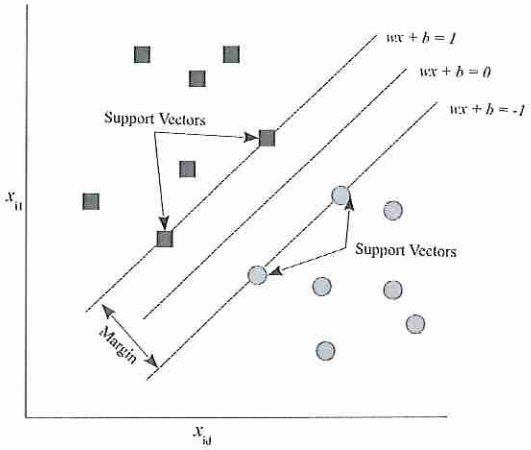

Fig. 3: A SVM model example [13].

- **Random Forest** [14] relies on two core principles: (i) the creation of hundreds of decision trees and their combination into a single model; and (ii) the ~~closing~~ decision is based on the ruling of the majority of the forming trees.

2) Feature Selection Methods: Feature selection is the process of choosing a subset of features that more contribute to the accuracy of a predictive model. Following a brief description of three common feature selection methods [15]:

- **Information Gain (IG)**: this method measures the number of bits of information obtained for category prediction by knowing the presence or absence of a feature in a dataset.
- **Chi-square (CHI)**: this method measures the lack of independence between a feature f and category c_i and can be compared to the chi-square distribution with one degree of freedom to judge extremeness.
- **Correlation Coefficient (CC)**: this method defines the correlation coefficient of feature f with a category c_i .

3) Evaluation measures: Accuracy, precision, recall, and F-measure are four measures commonly utilized to evaluate the performance of prediction models [16]. The computation of the values of these measures are based on a *confusion matrix* [13], which represents the number of true/false positives, and the number of true/false negatives for each instance class value when making a prediction. Following a brief description of each measure [17]:

- **Accuracy** is the percentage of correctly classified observations among all observations [17]:

$$Accuracy = \frac{TP + TN}{P + N} \quad (1)$$

Where P is total of positive class instances, N is total of negative class instances, TP is the number of true positives, and TN is the number of true negatives.

- **Precision** is the percentage of correctly classified observations among all observations that were assigned to the class by the classifier. It can be thought of as a measure of classifier exactness. A low precision can also indicate a large number of false positives. More formally recall is defined by the next expression [17]:

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

Where TP is the number of true positives, and FP is the number of false positives.

- **Recall** of a classification method can be defined as the percentage of correctly classified observations among all observations belonging to that class. It can be thought of as a measure of a classifiers completeness. A low recall indicates many false negatives in testing classification step. More formally recall is defined by the next expression [17]:

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

Where TP is the number of true positives, and FN is the number of false negatives.

- **F-measure** is the weighted harmonic mean of the precision and recall [16], [17]. F-measure can be calculated using the next formula:

$$F\text{-}measure = \frac{2 \times (precision \times recall)}{(precision + recall)} \quad (4)$$

Receiving Operating Characteristics (ROC) is an alternative measure to evaluate binary classifiers. A ROC curve [17] is a bidimensional chart, in which X represents false positives values, and Y represents true positives values. The Area Under ROC Curve (AUC), ranging between 0 and 1, is used to assess the performance of ML algorithms. An algorithm outperforms another one if its AUC value is closer to 1.

4) Sampling methods: Typically, there is only one dataset which must be used to train and test a predictive model. Assessing performance of this model using the same data employed during training may yield misleading optimistic estimations [17]. Resampling methods help to obtain more reliable predictive estimates splitting the entire dataset into training data and test data. The first subset is used to generate and adjust the model, while the second subset simulates new data to evaluate the predictor. Following a brief description of three traditional resampling methods used in machine learning [18]:

- **Holdout** splits a dataset into a ratio of p for training and $(1 - p)$ for testing.
- **Cross-Validation (CV)** divides a dataset into k folds. At in each iteration, it reserves a different fold for testing and uses all the others for training.
- **Bootstrap** generates r training subsets from the original dataset. It randomly samples instances with replacement from this set. Unselected cases make up the test subset. The result is the average performance observed in each test subset.

5) Statistical tests: Many scenarios require running more than one ML algorithm to pick up that generate the best predictive model. Experiments with this algorithms can produce very similar performances and differences between evaluation measures may not be significant. Therefore, deciding whether one algorithm is better than another examining only these outcomes may be difficult and is not recommended [18]. One needs to conduct statistical tests for reliable performance comparison between models under investigation. The primary purpose of statistical testing is to gather evidence about which results yielded by evaluation measures represent a general behavior of classifiers [17]. Following a brief description of four common statistical tests:

- **T-Test** [17] is a test of the null hypothesis that can be used to assess whether the means of two groups are statistically different from each other.
- **Wilcoxon signed-rank test** [19] is a test of the null hypothesis that can be used to determine whether two dependent samples, selected from the population, having the same distribution.
- **Proportion test** [20] is a test of the null hypothesis that can be used to assess whether or not a sample from a population represents the true proportion of the entire population.
- **Shapiro Wilk test** [20] is a test of the null hypothesis that can be used to test whether a sample $x_1, ..., x_n$ came from a normal distribution of the population.

C. Text Mining

The common ML algorithms cannot directly process unstructured text (e.g., summary and description fields from bug report form). Therefore, during a preprocessing step, these unstructured texts are converted into a more manageable representation. Typically, they are represented by feature vectors, points of an n-dimensional space. Text mining is the process of converting unstructured text into a structure suited to analysis [16]. It is composed of three primary activities [13]:

- **Tokenization** is the action to parsing a character stream into a sequence of tokens by splitting the stream at delimiters. A token is a block of text or a string of characters (without delimiters such as spaces and punctuation) which is a useful portion of the unstructured data.

- **Stop words removal** eliminates commonly used words that do not provide relevant information to a particular context, including prepositions, conjunctions, articles, common verbs, nouns, pronouns, adverbs, and adjectives.
- **Stemming** is the process of reducing or normalizing inflected (or sometimes derived) words to their word stem, base form—generally a written word form (e.g., "working" and "worked" into "work").

Two of the most traditional ways of representing a document relies on the use of a bag of words (unigrams) or a bag of bigrams (when two terms appear consecutively, one after the other) [16]. In this approach all terms represent features, and thus the dimension of the feature space is equal to the number of different terms in all texts. Methods for assigning weights to features may vary. The simplest is to assign binary values representing the presence or absence of the term in each text. Term Frequency (TF), another type of quantification scheme, considers the number of times in which the term appears in each text. Term Frequency-Inverse Document Frequency (TD-IDF), a more complex type of scheme, takes into account the frequencies of the term in each text, and in the whole collection of all texts. The importance of a term in this scheme is inversely proportional to the frequency of the term in the collection of all texts.

III. RELATED WORK

To the extent of our knowledge, only two papers [1], [21] have reviewed the literature about bug report severity prediction. Cavalcanti et al. [1] performed a review of 142 papers that investigated challenges and opportunities for software change or bug report request. Just seven of them are related to change request prioritization, which is defined in Bugzilla by two fields: priority and severity. Four out of them addressed bug report severity prediction. Only two papers (Lamkanfi et al. [4] and Lamkanfi et al. [5]), however, addressed severity prediction on FLOSS projects. This mapping review shows that all of seven papers used some Information Retrieval Model: one paper used the vector space representation (binary and term count), and all seven used TF-IDF. Also, it shows that all papers implement learning techniques such as SVM (4 out of 7), Decision tree (2 out of 7), k-NN (2 out of 7), Naïve Bayes (3 out of 7), and Naïve Bayes Multinomial (1 out of 7).

The review presented by Uddin et al. [21], in turn, surveyed published papers in bug prioritization. The authors reviewed and analyzed in depth 32 distinct papers published between 2003 and 2015. The aim of that analysis was "to summarize the existing work on bug prioritization and some problems in working with bug prioritization". That work categorizes research initiatives according to ML algorithms, evaluation measures, data sets, researchers, and publication venue. It can be worth to note that the authors investigated only eight papers about predicting of severity level. In contrast, the current mapping review investigated in depth 27 papers about bug report severity prediction published from 2010 to 2017.

Although these reviews present relevant results for research in the theme, they have one essential limitation. There is no explicit focus on bug report severity prediction. In fact, these papers perform a brief review of this topic by addressing mainly bug report priority prediction. The current review has a broader goal of mapping studies addressing many research questions and concepts related to bug report severity prediction. Furthermore, this review considers relevant aspects (e.g., sampling techniques and statistical tests) in the characterization of papers data than previous surveys.

IV. RESEARCH METHOD

This section describes the research method used in this mapping review for identifying and analyzing relevant papers. It based on the software engineering systematic literature review guidelines and recommendations proposed by Kitchenham et al. [6], Brhel et al. [22], Souza et al. [23], and Petersen et al. [7]. These authors suggest a process with three main steps: planning, conducting, and reporting. In the planning step, they recommend defining a review protocol that includes a set of research questions, inclusion and exclusion criteria, sources of papers, search string, and mapping procedures. In the conducting step, they recommend selecting and retrieving papers for data extraction. Finally, in the reporting step, they recommend analyzing the results used to address research questions defined before.

A. Research questions

The main goal of this mapping review is to provide a current status of the research on bug report severity prediction in FLOSS. To ensure an unbiased selection process, this review addresses the following research questions.

- **RQ1. When and where have the studies been published?** This research question gives to the researcher a perception whether the topic of this mapping review seems to be broad and new, providing an overview about where and when papers were published.
- **RQ2. What FLOSS are the most used as report sources in experiments for bug report severity prediction?** This research question names the FLOSS used as report source in problems of bug report severity prediction. This overview can be useful for researchers that intend to accomplish new initiatives in this area, and can also motivate adoption of new FLOSS in future research to bridge existing gaps.
- **RQ3. Was bug report severity prediction most addressed as either a fine-grained label or coarse-grained label prediction problem?** This research question investigates whether bug report severity prediction was handled as a fine-grained label (i.e., multi-label) or as a coarse-grained label (i.e., two-label or three-label) prediction problem. Understanding each solution provided for each prediction problem type is essential.

RQ4. What are the most common features used for bug report severity prediction? This question investigates features used for bug report severity prediction in FLOSS. It can help to map which features have been considered more effective for this prediction problem.

RQ5. What are the most common features selection methods used for bug report severity prediction? This question complements the previous one, looking for feature selection methods employed in the papers for bug report severity prediction in FLOSS. Also, this RQ allows for identifying which feature selection approaches have been considered more suitable for this prediction problem.

RQ6. What are the most used text mining or information retrieval methods for bug report severity prediction? This research question indicates the main text mining approaches used to extract features from textual fields of bug reports. It can guide researchers in the task of selecting text mining methods more suitable to be applied in bug report severity prediction or similar problems.

RQ7. What are the most used machine learning algorithms for bug report severity prediction? This research question aims to identify the main ML algorithms, which have been adopted for bug report severity prediction in FLOSS. Also, this RQ can be useful for any researcher that intends to accomplish further initiatives in this area, as well as to guide him in the search of other algorithms to improve state of the art.

RQ8. What are the measures typically used to evaluate ML algorithms performance for bug report severity prediction? This research question aims to find out which measures are used to evaluate ML algorithms performance. It can be helpful to identify the more appropriate measures to evaluate ML algorithms performance for bug report severity prediction.

RQ9. Which sampling techniques are applied most frequently to generate more reliable predictive performance estimates in severity prediction of a bug report? This question complements the previous one, investigating sampling techniques, which have been used to generate more reliable and accurate ML models. It can be useful to analyze sampling techniques, which might be more suitable to improve machine learning algorithms performance for bug report severity prediction.

RQ10. Which statistical tests were used to compare the performance between two or more ML algorithms for bug report severity prediction? The answer to this research question can be useful to identify which statistical tests were used in the context of bug report severity prediction. In addition, it allows grasping existing protocols for the comparison between two or more ML algorithms performance using statistical significance tests.

RQ11. Which software tools were used to run experiments for bug report severity prediction? This research question highlights the software tools most used to run experiments for bug reports severity prediction in FLOSS projects. It is useful to provide information for researchers and practitioners about technologies that could be used in their experiments.

RQ12. Which solution types were proposed for the problem of bug report severity prediction? This research question examines whether the most common solutions proposed in the selected studies are either online or off-line. This is important to separate the solutions that can be applied only in an experimental environment (off-line) from those that also can be deployed in a real scenario (online).

B. Paper selection

This section describes the selection process carried out through this systematic mapping review. It comprises four steps: (i) terms and searching string; (ii) sources for searching; (iii) inclusion and exclusion criteria; and (iv) data storage procedures.

1) Terms and search string: The base string was constructed from three main search terms related to three distinct knowledge areas: "open source project", "bug report" and "severity predict." To build the search string, terms were combined with "AND" connectors. The search string syntax was adapted according to particularities of each source (e.g., wildcards, connectors, apostrophes, and quotation marks) before it has been applied on three metadata papers: title, abstract, and keywords. Table II exhibits terms and final search string used in this systematic mapping review.

TABLE II: Search string.

Area	Search term
Bug report	"bug report"
Open source	"open source software"
Severity prediction	"severity predict"
Search string:	"open source software" AND "severity predict" AND "bug report"

2) Sources: To accomplish this systematic mapping review, four electronic databases and one popular search engines were selected, as recommended by Kitchenham et al. [6]. Table III shows each selected search sources, as well as the search date, and the period covered by the search.

TABLE III: Search sources.

Source	Electronic address	Type	Search Date	Years covered
ACM	http://dl.acm.og	Digital library	Dec,18	2010 - 2017
IEEE	http://ieeexplore.ieee.org	Digital library	Dec,18	2010 - 2017
Google Scholar	http://www.scholar.google.com	Search engine	Dec,18	2010 - 2017
Science Direct	http://www.sciencedirect.com	Digital library	Dec,18	2010 - 2017
SpringLink	http://www.springerlink.com	Digital library	Dec,18	2010 - 2017

3) Inclusion and exclusion criteria: The inclusion criteria below allow the identification of papers on the existing literature.

IC_1. The paper discusses bug report severity prediction in FLOSS projects.

On the contrary, the following exclusion criteria allows excluding all papers that satisfy any of them.

EC_1. The paper does not have an abstract;
EC_2. The paper is just published as an abstract;
EC_3. The paper is written in a language other than English;
EC_4. The paper is not a primary paper (e.g., keynotes);
EC_5. The paper is not accessible on the Web;

4) Data storage: The data extracted in the searching phase were stored into a spreadsheet that recorded all relevant details from selected papers. This spreadsheet supported classification and analysis procedures throughout this systematic mapping review.

5) Assessment: According to Kitchenham et al. [6], the consistency of the protocol utilized in a systematic mapping should be reviewed to confirm that: "the search strings, constructed interactively, derived from the research questions"; "the data to be extracted will properly address the research question(s)"; "the data analysis procedure is appropriate to answer the research questions". In this research, the first author is a Ph.D. candidate running the experiments. The second and third authors conducted the review process.

C. Data extraction and synthesis

Figure 4 illustrates the four-stage selection process carried out in the current mapping review. In each stage, the sample size was reduced based on the inclusion and exclusion criteria. In the first stage, relevant papers were retrieved by querying the databases with the search string presented in Table II. All database queries were issued in December 2017. This yielded a total of 54 initial papers: 13 from **IEEE Xplore**, 5 from **Science Direct**, 17 from **ACM Digital Library**, and 19 from **SpringerLink**. After removing four duplicates, we reduced (around 7%) the initial set of 50 papers. In the second stage, inclusion and exclusion criteria were applied over title, abstract, and keywords, reaching a set of 18 papers (reduction around 64%): 32 papers were rejected for not satisfying IC_1 (The paper discusses severity prediction of bug reports in FLOSS projects). In the third stage, exclusion criteria were applied considering the full text. However, no paper was rejected by taking into account these criteria.

In the fourth stage, the Snowballing (search for references) [6] activity was conducted, which resulted in 12 additional papers. After applying selection criteria over title, abstract, and keywords, 11 papers remained (reduction of 8.3% over the papers selected by snowballing). For these papers, selection criteria were applied considering the full text, and nine papers remained (reduction of approximately 18.2% over the 11 previously selected papers); EC_3 criterion eliminated one paper (the paper is written in a language other than English); EC_6 eliminated one more paper (the paper is not accessible on the Web), and another one was rejected for not satisfying IC_1. The selection phase ended up with 27 papers to be analyzed (18 from the sources plus nine from snowballing). Table IV shows the bibliographic reference of selected papers and an reference identifier (ID) for each paper. Throughout the remainder of this text, these identifiers will be used to refer to the corresponding paper.

D. Categorization scheme

Petersen et al. [7] suggest the definition of a categorization scheme for conducting a systematic mapping analysis. The categories defined for this mapping review were based on two approaches: (i) categories available in the literature and (ii) taking into account the selected papers. The next sections outline the categories considered in this mapping review.

1) FLOSS software type (RQ_2): This scheme organizes FLOSS projects by software type. Based on the taxonomy suggested by Pressman [46], FLOSS in studies belongs to three categories:

a. **Application software**: This category includes computer programs designed to solve a specific problem or business need for end users.
b. **System software**: This category includes collections of computer programs (operating systems and utilities) required to run and maintain a computer system.
c. **Programming tools**: This category includes computer programs that aid software developers in creating, debugging, maintain or perform any development-specific task.

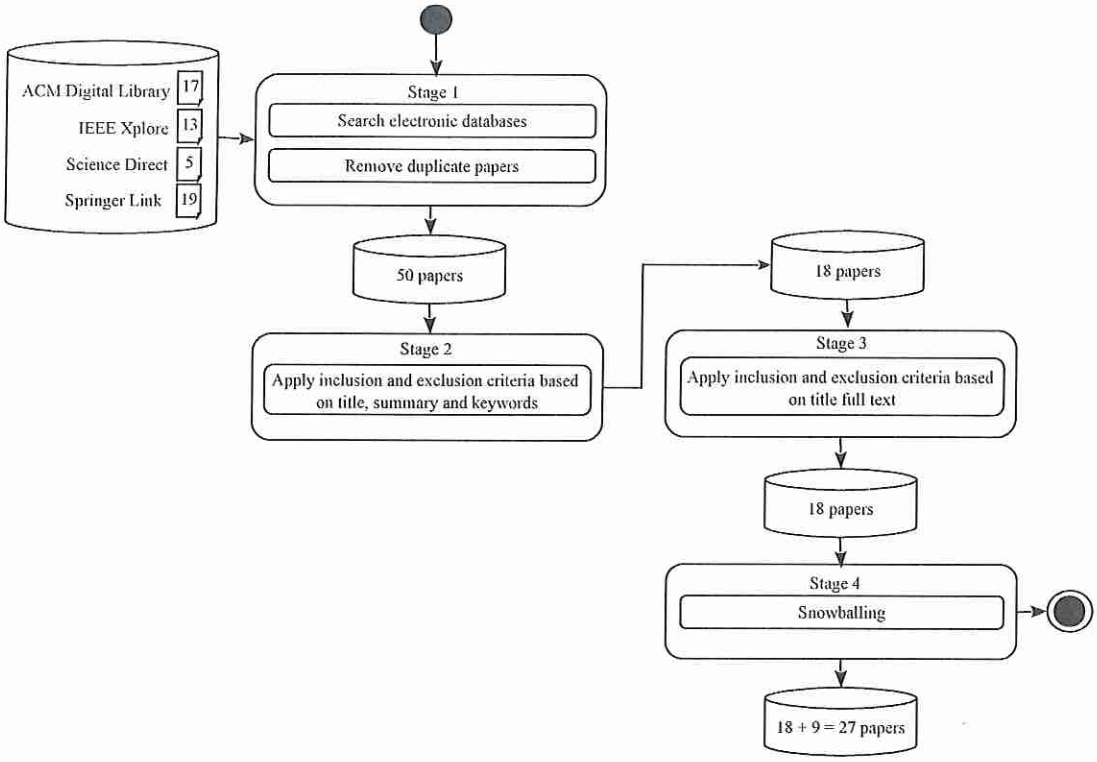

Fig. 4: Diagram of the four-stage paper selection process.

2) Severity prediction problem (RQ_3): This scheme drills down severity prediction problems, based on studying of selected papers, in five categories:

a. **Severe or Non-Severe (SNS)**: This category comprises the prediction problems whose the predicted severity level might be severe or non-severe. In Bugzilla, for example, the severe class includes blocker, critical, and major; the non-severe class includes minor and trivial levels. The predictors of this problem category do not take into account the default severity level.

b. **Severe or Non-Severe With Default Class (SNSWD)**: This category is quite similar to SNS. However, the predictors consider the default severity level as severe or non-severe, or yet as another class.

c. **Multiple Classes (MC)**: This category comprises prediction problems whose predictors treat each severity level as a different class. Likewise that in SNS, the predictors do not take into account the default severity level.

d. **Multiple Classes With Default Class (MCWD)**: This category is similar to MC. However, the predictors consider the default severity level as a regular class.

e. **Blocking or Non-Blocking (BNB)**: The prediction problem whose predictors classify a bug report severity level into blocking or non-blocking class. A blocking bug is software defect that prevents other defects from being fixed [9].

3) Feature data types (RQ_4): This scheme groups the features used for bug reports severity prediction into three categories:

a. **Qualitative categorical**: This category encompasses features that contain an unordered list of values. For instance, bug report platform attribute (e.g., "Windows", "Linux", "Android").

b. **Qualitative ordinal**: This category encompasses features that contains an ordered list of values. For instance, bug report priority attribute (e.g., P1, P2, P3).

c. **Quantitative discrete**: This category encompasses features that contain a finite or an infinite number of values that can be counted, such as the bug fixed time attribute.

d. **Quantitative continuous**: This category encompasses features that contain an infinite number of values that can be measured, such as the summary weight attribute.

e. **Unstructured text**: This category encompasses features that either does not have a pre-defined data model or is not organized in a pre-defined manner. For instance, the bug report description attribute which usually includes free format texts.

4) Feature selection methods (RQ_5): This scheme organizes the feature selection methods used for bug reports severity prediction into three categories, according to the taxonomy described in Guyon and Elisseeff [47]:

a. **Filter**: This category comprises methods that assign a score to the features, and use this value as a selection criterion to pick the top-scoring ones.

TABLE IV: Selected papers

ID	Bibliographic reference
[4]	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)*, May 2010, pp. 1–10.
[5]	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*, March 2011, pp. 249–258.
[3]	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*, Oct 2012, pp. 215–224.
[24]	C. Z. Yang, C. C. Hou, W. C. Kao, and I. X. Chen, "An empirical study on improving severity prediction of defect reports using feature selection," in *2012 19th Asia-Pacific Software Engineering Conference*, vol. 1, Dec 2012, pp. 240–249.
[25]	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, no. 2, pp. 32–59, 2012.
[26]	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*, June 2014, pp. 216–219.
[27]	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*, July 2014, pp. 97–106.
[9]	H. Valdivia Garcia and E. Shihab, "Characterizing and predicting blocking bugs in open source projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 72–81.
[28]	M. Sharma, M. Kumari, R. K. Singh, and V. B. Singh, "Multiattribute based machine learning models for severity prediction in cross project context," in *Computational Science and Its Applications – ICCSA 2014*, B. Murgante, S. Misra, A. M. A. C. Rocha, C. Torre, J. G. Rocha, M. I. Falcão, D. Taniar, B. O. Apduhan, and O. Gervasi, Eds. Cham: Springer International Publishing, 2014, pp. 227–241.
[29]	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*, Aug 2014, pp. 269–276.
[30]	R. K. Saha, J. Lawall, S. Khurshid, and D. E. Perry, "Are these bugs really "normal"?" in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, May 2015, pp. 258–268.
[8]	T. Zhang, G. Yang, B. Lee, and A. T. S. Chan, "Predicting severity of bug report by mining bug repository with concept profile," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ser. SAC '15. New York, NY, USA: ACM, 2015, pp. 1553–1558.
[31]	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, proceedings of the 4th International Conference on Eco-friendly Computing and Communication Systems.
[32]	X. Xia, D. Lo, E. Shihab, X. Wang, and X. Yang, "Elblocker: Predicting blocking bugs with ensemble imbalance learning," *Information and Software Technology*, vol. 61, pp. 93 – 106, 2015.
[33]	S. Gujral, G. Sharma, S. Sharma, and Diksha, "Classifying bug severity using dictionary based approach," in *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, Feb 2015, pp. 599–602.
[34]	M. N. Pushpalatha and M. Mrunalini, "Predicting the severity of bug reports using classification algorithms," in *2016 International Conference on Circuits, Controls, Communications and Computing (I4C)*, Oct 2016, pp. 1–4.
[35]	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)*, April 2016, pp. 92–95.
[36]	K. K. Sabor, M. Hamdaqa, and A. Hamou-Lhadj, "Automatic prediction of the severity of bugs using stack traces," in *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering*, ser. CASCON '16. Riverton, NJ, USA: IBM Corp., 2016, pp. 96–105.
[37]	Y. Tian, N. Ali, D. Lo, and A. E. Hassan, "On the unreliability of bug severity data," *Empirical Softw. Engg.*, vol. 21, no. 6, pp. 2298–2323, dec 2016.
[38]	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.
[39]	K. Jin, A. Dashbalbar, G. Yang, J.-W. Lee, and B. Lee, "Bug severity prediction by classifying normal bugs with text and meta-field information," *Advanced Science and Technology Letters*, vol. 129, pp. 19–24, 2016.
[40]	T. Chocikiwong and P. Vateekul, "Improve accuracy of defect severity categorization using semi-supervised approach on imbalanced data sets," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, 2016.
[41]	K. Jin, A. Dashbalbar, G. Yang, and B. Lee, "Improving predictions about bug severity by utilizing bugs classified as normal," *Contemporary Engineering Science*, vol. 9, pp. 933–942, 2016.
[42]	K. Jin, E. C. Lee, A. Dashbalbar, J. Lee, and B. Lee, "Utilizing feature based classification and textual information of bug reports for severity prediction," *Information*, vol. 19, no. 2, pp. 651–659, Feb 2016.
[43]	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*, ser. SAC '17. New York, NY, USA: ACM, 2017, pp. 1280–1287.
[44]	V. B. Singh, S. Misra, and M. Sharma, "Bug severity assessment in cross project context and identifying training candidates," *Journal of Information and Knowledge Management*, vol. 16, no. 01, p. 1750005, 2017.
[45]	N. K. S. Roy and B. Rossi, "Cost-sensitive strategies for data imbalance in bug severity classification: Experimental results," in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Aug 2017, pp. 426–429.

Methods within this category

b. **Wrapper**: This category comprises methods that prepare, evaluates and compares feature combinations, and selects the best one. It considers the feature selection as a search problem.

OK c. **Embedded**: This category comprises methods that select which features best contribute to the accuracy while the ML algorithm creates the predicting model.

5) Text mining feature representations (RQ_6): This scheme organizes the techniques for text mining feature representations used for bug report severity prediction in four categories. The following categories were employed in this systematic mapping review according to [16]:

a. **Character**: This category includes techniques that represent features as individual component-level letters, numerals, special characters, and spaces are the building blocks of higher-level semantic features such as words, terms, and concepts.
b. **Word**: This category includes techniques that represent features as specific words selected directly from a "native" document are at what might be described as the basic level of semantic richness.
c. **Term**: This category includes techniques that represent features as single words an multiword phrases selected directly from a corpus of a native document using term-extraction methodologies.
d. **Concept**: This category includes techniques that represent features generated for a document employing manual, statistical, rule-based, or hybrid categorization methodologies.

6) ML algorithms categories (RQ_7): This scheme groups ML algorithms used in bug report severity prediction in five categories. The following categories were based on taxonomy proposed by Facelli et al. [18]:

a. **Distance-based**: This category comprises algorithms that use the proximity between data to generate their predictions.
b. **Probabilistic-based**: This category comprises algorithms that make their predictions based on the Bayes's theorem.
c. **Searching-based**: This category comprises algorithms that rely on searching for a solution in space to generate their predictions.
d. **Optimization-based**: This category comprises algorithms whose predictions are based on an optimization function.
e. **Ensemble-method**: This category comprises algorithms that combine or aggregate results of different base classifiers to make a prediction.

7) Evaluation measures categories (RQ_8): This scheme organizes evaluation measures used in selected papers by categories. According to Japkowicz and Shah [17], there are six categories:.

a. **Single class focus**: This category comprises measures based on confusion matrix information whose focus is on a single class of interest.
b. **Multi-class focus**: This category comprises measures based on confusion matrix information whose focus is on all the classes of the problem domain.
c. **Graphical measure**: This category comprises measures based on confusion matrix in conjunction with extra information. They enable visualization of the classifier performance under different skew ratios and class priors distribution and are indicated for scoring classifiers.
d. **Summary statistics**: This category comprises measures based on confusion matrix in conjunction with extra information. They enable to quantify the comparative analysis between classifiers and are indicated for scoring classifiers.
e. **Distance/error-measure**: This category comprises measures based on confusion matrix in conjunction with extra information. They measure the distance of an instance's predicted class label to its actual label and are recommended for continuous and probabilistic classifiers.
f. **Information theoretic measures**: This category comprises measures based on confusion matrix in conjunction with extra information. They reward a classifier upon correct classification relative to the (typically empirical) prior to the data and are indicated for continuous and probabilistic classifiers.

8) Sampling techniques (RQ_9): This scheme organizes the sampling techniques used for bug report severity prediction into three categories. Such categories according to Japkowicz and Shah [17] include:

a. **No re-sampling**: This category encompasses techniques that consist of testing the algorithm on a large set of unseen data.
b. **Simple re-sampling**: This category encompasses techniques that tend to use each data point for testing only once.
c. **Multiple re-sampling**: This category encompasses techniques that tend to use each data point for testing more than once.

9) Statistical Tests (RQ_{10}): This scheme groups statistical tests commonly used for bug report severity prediction in three categories. Such categories employed in this mapping review according to Japkowicz and Shah [17] include:

a. **Parametric**: This category comprises methods that make strong assumptions about the distribution of the population
b. **Non-parametric**: This category comprises methods that do not make strong assumptions about the distribution of the population
c. **Parametric and non-parametric**: This comprises methods that are both parametric and non-parametric

10) Experiment software tools (RQ_{11}): This scheme groups the software tools used in experiments for bug reports severity prediction in two well-known and popular categories:

a. **Free/Libre Open Source Software (FLOSS)**: This category includes software that can be freely used, modified, and redistributed.
b. **Closed Source Software (CSS)**: This category includes software that is owned by an individual or a company whose source code is not shared with the public for anyone to look at or change.

E. Limitations of this mapping

To interpret the implications of our results adequately, one needs to consider the following common limitations of all mapping reviews:

- **Mapping review completeness can never be guaranteed [22]**: Although this mapping review has observed a strict research protocol to ensure the relatively whole population of the relevant literature, some important papers might be missed.
- **Terminology problems in search string may lead to miss some primary studies [23]**: The terminology applied in the database query is normally accepted and used within the scientific community. Nevertheless, different terms may be used to retrieve the same relevant information.
- **Subjective evaluation of inclusion and exclusion criteria may cause misinterpretation [22]**. Explicit criteria have been defined for assessing the relevance of selected papers IV-B. However, the evaluation was based on the perception and experience of the authors. Different people might have other views regarding the relevance of these papers.

V. Results

This section summarizes the results of this mapping review. The answer to each research question (RQ1 to RQ12) is presented in tables and charts. The organization of extracted data followed the criteria defined in Section IV.

A. When and where have the studies been published? (RQ1)

Figure 5a demonstrates that research on bug report severity prediction in FLOSS is recent and active with a vast number of papers (22 out of 27 $\approx$ 81%) published from 2014, while Figure 5b shows that the conferences were the main venue used to publicize the papers. Journals and conferences on information technology, mining software repositories, and software engineering seem to be more receptive to papers of bug report severity prediction (Table V).

TABLE V: Papers publication sources.

Paper source	Type	Reference
ACM Symposium on Applied Computing	Conference	[8]
Advanced Science and Technology Letters & Journal	Journal	[39]
Asia-Pacific Software Engineering Conference	Conference	[24]
Computational Science and Its Applications (ICCSA)	Conference	[28]
Computer Software and Applications Conference	Conference	[27]
Contemporary Engineering Sciences	Journal	[41]
Empirical Software Engineering	Journal	[37]
Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)	Conference	[33]
Information and Software Technology	Journal	[32]
International Conference on Circuits, Controls, Communications and Computing (I4C)	Conference	[34]
International Conference on Computer Science and Software Engineering	Conference	[36]
International Conference on Information and Communication Systems (ICICS)	Conference	[35]
International Conference on Software Engineering and Service Science	Conference	[26]
International Information Institute	Journal	[42]
International Journal of Open Source Software and Process(IJOSSP)	Journal	[25]
International MultiConference of Engineers and Computer Scientists	Conference	[40]
Journal of Information & Knowledge Management	Journal	[44]
Journal of Systems and Software	Journal	[38]
Procedia Computer Science	Journal	[31]
Software Engineering and Advanced Applications (SEAA)	Conference	[29], [45]
Symposium on Applied Computing (SAC)	Conference	[43]
Working Conference on Mining Software Repositories (MSR)	Conference	[4], [5], [9], [30]
Working Conference on Reverse Engineering	Conference	[3]

B. What FLOSS are the most used as experiment target for bug report severity prediction (RQ2)?

Table VI demonstrates that the most papers concentrated their focus on five FLOSS: Eclipse (≈ 92%), Mozilla (≈ 70%), Openoffice (≈ 18%), Netbeans (≈ 14%), and Gnome (≈ 11%). The detailing of each FLOSS including description, category, BTS, and URL is presented in Table VII.

TABLE VI: Paper distribution by FLOSS.

Project	References	Total
Eclipse	[4], [5], [3], [24], [25], [27], [26], [9], [29], [30], [8], [31], [32], [33], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [5], [3], [24], [25], [27], [26], [9], [29], [30], [8], [31], [32], [33], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45]	25
Mozilla	[4], [3], [24], [25], [27], [9], [28], [29], [8], [32], [34], [35], [37], [38], [39], [41], [42], [44], [45]	19
Openoffice	[3], [9], [32], [37], [38]	5
Netbeans	[27], [9], [32], [38]	4
Gnome	[4], [5], [25]	3
Chromium	[9], [32]	2
Freedesktop	[9], [32]	2
GCC	[38]	1
Hibernate	[45]	1
Spring	[45]	1
Android	[43]	1
JBoss	[43]	1
Mongo-db	[45]	1
WineHQ	[34]	1

Figure 5c shows that the most papers worked with FLOSS of Programming Tool (≈ 92%) and Application Software (≈ 74%) categories. Moreover, it demonstrates that 17 out of 27 (≈ 62%) papers worked with both categories. Figure 5d highlights that all papers treated bug reports extracted from Bugzilla, and a just few papers from Google (3 out of 27 - ≈ 11%) and Jira (2 out of 27 ≈ 7%). Only one paper (Yang et al. [43]) investigated bug reports from three BTS.

TABLE VII: FLOSS investigated in papers.

Project	Description	Category	BTS	URL (As of July 3, 2018)
Android	Mobile operating system	System software	Google	https://issuetracker.google.com
Chrominum	Web browser	Application software	Google	https://www.chromium.org/issue-tracking
Eclipse	Integrated Development Environment(IDE)	Programming tool	Bugzilla	https://bugs.eclipse.org/bugs/
FreeDesktop	Base platform for desktop software on Linux and UNIX	Programing Tool	Bugzilla	https://bugs.freedesktop.org/
GCC	C compiler	Programming tool	Bugzilla	https://gcc.gnu.org/bugzilla/
Gnome	Desktop environment based on X Windows System	Application Software	Bugzilla	https://gcc.gnu.org/bugzilla/
Hibernate	Object Relation Mapper(ORM) framework	Programming tool	Jira	https://hibernate.atlassian.net
Jboss	Application server	System software	Jira	https://issues.jboss.org/s
Mongo-db	No-sql database	System software	Jira	https://jira.mongodb.org/
Mozilla	Internet tools	Application software	Bugzilla	https://bugzilla.mozilla.org/
Netbeans	Integrated Development Environment(IDE)	Programming tool	Bugzilla	https://netbeans.org/bugzilla/
OpenOffice	Office suite	Application software	Bugzilla	https://bz.apache.org/ooo/
Spring	JEE Framework	Programming tool	Jira	https://jira.spring.io
WineHQ	Compatibility layer	System software	Bugzilla	https://bugs.winehq.org/

C. Was bug report severity prediction most addressed as either a fine-grained label or coarse-grained label prediction problem (RQ3)?

Table VIII demonstrates that a little more than half of the papers (≈ 52%) addressed the bug report severity prediction as a coarse-grained problem and ≈ 48% of papers as a fine-grained problem. Figure 6a shows that the most papers (10 out of 27 ≈ 37%) addressed the severity prediction as a problem of SNS category. Moreover, 8 out of 27 (≈ 20%) papers addressed it a problem of MC category and 5 out of 27 (≈ 18.5%) addressed a problem of MCWD category. A few papers addressed a BNB (2 out of 27 ≈ 7%) or an SNSWD (2 out of 27 ≈ 7%) problem. No paper addressed more than a category of problem.

veja se as minhas alterações fazem sentido

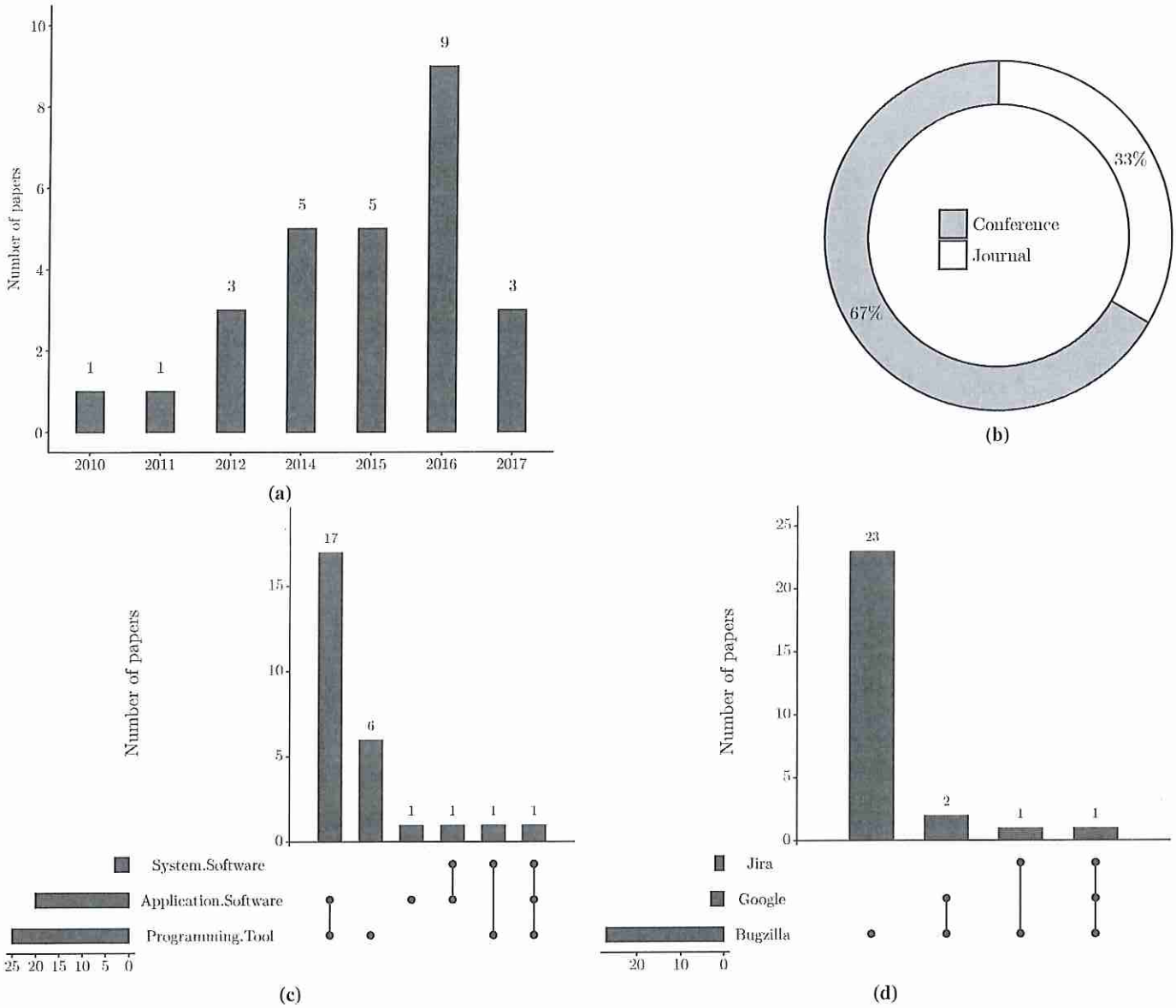

Fig. 5: (a) Paper distribution by year, (b) % of papers by source, (c) paper distribution by FLOSS (d) paper distribution by BTS.

TABLE VIII: Paper distribution by prediction problem.

Problem	ReferenceS	Total
Coarse-grained	[4], [5], [24], [26], [29], [9], [31], [33], [32], [35], [39], [41], [30], [42]	14
Fine-grained	[3], [25], [27], [28], [8], [34], [36], [37], [38], [40], [44], [43], [45]	13

D. What are the most common features used for bug report severity prediction (RQ4)?

Table IX describes the features used for bug report severity prediction. Each feature in this table has a description, a category, an indicator of if the feature is computed from others, and an indicator of if the feature is available in Bugzilla, Jira, and Google.

As shown in Table X, most used features were summary (≈ 74%) and description (≈ 70.37%). Besides, a significant number of papers also reported using the features product (≈ 37%) and component (≈ 33%). Regarding data type of the features, unstructured text (26 out of 27 ≈ 96%) was the most used data type in the papers (Figure 6b). Only one paper (Meera et al. [28]) reported ~~using~~ features belongs to each data type presented in the chart.

TABLE IX: Feature descriptions.

Feature	Description	Category	Calculated?	Bugzilla	Jira	Google
Attachments	Files (e.g., test cases or patches) attached to bugs.	Qualitative Categorical	No	Yes	No	Yes
Bug fix time	Time to fix a bug (Last Resolved Time - Opened Time).	Quantitative Continuous	Yes	Yes	Yes	Yes
Bug id	Bug report identifier.	Quantitative Discrete	No	Yes	Yes	Yes
CC list	A list of people who get mail when the bugs changes.	Qualitative Categorical	No	Yes	No	Yes
Comment	Textual content appearing in the comments of bug report.	Unstructured Text	Yes	Yes	Yes	Yes
Comment size	The number of word of all comments of a bug.	Quantitative Discrete	Yes	Yes	Yes	Yes
Complexity	Bug level of complexity based on bug fix time.	Qualitative Ordinal	Yes	Yes	Yes	Yes
Component	Each product is divided into different components (e.g., Core, Editor, UI, etc).	Qualitative Categorical	No	Yes	Yes	Yes
Description	Textual content appearing in the description field of the bug report.	Unstructured Text	No	Yes	Yes	Yes
Description size	The number of words in the description.	Quantitative Discrete	Yes	Yes	Yes	Yes
Importance	The importance of a bug is a combination of its priority and severity.	Qualitative Discrete	No	Yes	No	No
Number in CC list	The number of developers in the CC list of the bug.	Quantitative Discrete	Yes	Yes	No	Yes
Number of comments	Number of comments added to a bug by users.	Quantitative Discrete	Yes	Yes	Yes	Yes
Number of dependents	Number of report bugs dependents	Quantitative Discrete	Yes	Yes	Yes	Yes
Number of duplicates	Number of duplicates of bug report	Quantitative Discrete	Yes	No	No	Yes
Platform	These indicate the computing environment where the bug was found (e.g., Windows, GNU/Linux, Android, etc).	Qualitative Categorical	No	Yes	Yes	Yes
Priority	Priority should normally be set by the managers, maintainers or developers who plan to work, not by the one filling the bug or by outside observers.	Qualitative Ordinal	No	Yes	Yes	Yes
Product	What general "area" the bug belongs to (e.g., Firefox, Thunderbird, Mailer, etc).	Qualitative Categorical	No	Yes	Yes	No
Report length	The content length of long description providing debugging information.	Quantitative Discrete	Yes	Yes	Yes	Yes
Reporter	The account of the user who created the bug report.	Qualitative Categorical	No	Yes	Yes	Yes
Reporter blocking experience	Counts the number of blocking bugs filed by reporter previous to this bug.	Quantitative Discrete	Yes	Yes	Yes	No
Reporter experience	Counts the number of previous bug reports filed by the reporter.	Quantitative Discrete	Yes	Yes	Yes	No
Reporter name	Name of the developer or user that files the bug	Qualitative Categorical	No	Yes	Yes	No
Severity	This indicates how severe the problem is – from blocker ("application unusable") to trivial ("minor cosmetic issue").	Qualitative Ordinal	No	Yes	Yes	No
Summary	A one-sentence summary of the problem.	Unstructured Text	No	Yes	Yes	Yes
Summary weight	Calculated using information gain criterion	Quantitative Continuous	Yes	Yes	Yes	Yes

E. What are the most common features selection methods employed for bug report severity prediction (RQ5)?

Table XI shows that few papers (≈ 40%) reported using a feature selection method for bug report severity prediction, and remarkably all papers only used methods belong to filter category.

F. What are the most used text mining methods for bug report severity prediction (RQ6)?

As shown in Table XII, ≈ 74% out of the papers used unigrams for feature vector extraction and ≈ 33% out of the papers used TF-IDF for feature vector weighting. Regarding similarity/dissimilarity functions, ≈ 7% reported using some function: one paper used BM25, one paper BM25ext, and other KL divergence.

Figure 6c shows that the most used text mining method (9 out of 15 ≈ 44%) belongs to term category. Besides, only one paper (Yang et al. [43]) reported using text mining belongs to each category presented in the chart.

[1]Topic Model [38] is a statistical model to identify "topics" from a collection of documents. Each topic includes the topic terms which appear in the documents and each document may belong to one ou more topics.

[2]BM25ext [38] is a extension of similarity function BM25, while BM25 computes similarity of a short document with a long document, BM25ext computes similarity between long documents.

[3]Kullback-Leibler(KL) [27] divergence measures the difference between two probability over the same variable x.

TABLE X: Paper distribution by features.

Feature	References	Total
Summary	[4], [5], [3], [24], [25], [27], [26], [28], [30], [31], [34], [35], [37], [38], [39], [41], [42], [43], [44], [45]	20
Description	[4], [3], [27], [26], [9], [29], [8], [32], [33], [34], [35], [37], [36], [38], [39], [41], [42], [43], [45]	19
Product	[3], [27], [9], [32], [34], [37], [38], [39], [41], [42]	10
Component	[3], [27], [9], [32], [34], [38], [39], [41], [42]	9
Severity	[9], [32], [39], [41], [42]	5
Priority	[27], [9], [28], [32]	4
Reporter	[39], [41], [42]	3
Description size	[9], [32]	2
Number in CC list	[9], [32]	2
Reporter blocking experience	[9], [32]	2
Reporter experience	[9], [32]	2
Reporter name	[9], [32]	2
Attachments	[26]	1
Bug fix time	[28]	1
Bug Id	[40]	1
CC list	[28]	1
Comment	[9]	1
Comment size	[32]	1
Complexity	[28]	1
Importance	[34]	1
Number of comments	[28]	1
Number of dependents	[28]	1
Number of duplicates	[28]	1
Platform	[32]	1
Report length	[26]	1
Summary weight	[28]	1

TABLE XI: Paper distribution by feature selection methods.

Method	Category	References	Total
Information Gain	Filter	[24], [25], [31], [37], [45]	5
Chi-Square	Filter	[24], [29], [31]	3
Correlation Coefficient	Filter	[24]	1

G. What are the most used machine learning algorithms for bug report severity prediction (RQ7)?

Table XIII demonstrates that most papers used k-NN and NB ($\approx$ 44%) and that NBM has also been used quite frequently ($\approx$ 40%). As shown in Figure 6d, the majority of the papers used a probabilistic-based algorithm (21 out of 27 $\approx$ 77%). The figure also shows that 15 out of 27 papers ($\approx$ 55%) used algorithms ~~belongs~~ belonging to a single category.

TABLE XII: Paper distribution by text mining models.

Technique	Category	References	Total
		Vector Extraction Models	
Unigrams	Character	[4], [5], [3], [24], [25], [27], [26], [29], [31], [33], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45]	20
Bigrams	Character	[3], [29], [37], [38]	4
Topic model[1]	Concept	[27], [8], [38]	3
		Vector Weighting Models	
TF-IDF	Term	[5], [25], [29], [8], [31], [33], [36], [44], [45]	9
TF	Term	[5], [24], [35]	3
Binary	Term	[5]	1
		Vector Similarity/Dissimilarity Functions	
BM25ext[2]	Term	[38]	1
BM25	Term	[3]	1
KL divergence[3]	Term	[27]	1

TABLE XIII: Papers distribution by ML algorithms

Algorithm	Category	References	Total
KNN	Distance	[5], [3], [27], [9], [28], [8], [33], [36], [37], [38], [40], [44]	12
NB	Probabilistic	[4], [5], [25], [27], [9], [28], [29], [30], [8], [35], [40], [44]	12
NBM[4]	Probabilistic	[5], [24], [26], [8], [31], [33], [37], [39], [41], [42], [43]	11
SVM	Otimization	[5], [25], [28], [37], [40], [44], [45]	7
Random Forest	Ensemble	[25], [9], [32], [35], [40]	5
C4.5[5]	Searching	[25], [9], [34]	3
Bagging	Ensemble	[32], [34]	2
Decision Tree	Searching	[37], [40]	2
AdaBoost	Ensemble	[35]	1
Functional Tree	Searching	[35]	1
Random Tree	Ensemble	[35]	1
RBF Networks	Optimization	[35]	1
RIPPER[6]	Other	[25]	1
Zero-R[7]	Other	[9]	1

H. What are the measures used to evaluate ML algorithms performance for bug report severity prediction (RQ8)?

The most of papers evaluated the ML model accuracy in bug report severity prediction using three based confusion matrix measures (Table XIV): precision (≈ 77%), recall (≈ 70%), and f-measure (≈ 66%). The majority of the used measures are

[4]Näive Bayes Multinomial (NBM) is similar to Näive Bayes. However, the output class is not only determined by presence or absence of term, but NBM also uses the number of occurrences of the terms to decide [5].

[5]C4.5 is an algorithm used to generate a decision tree which follows a greedy divide and conquer strategy in training step [9].

[6]Bagging Ensemble: use multiple learning algorithms to obtain better predictive performance that could be received from any of the constituent learning algorithms alone. Involves having each model in the ensemble vote with equal weight. To promote model variance, bagging trains each model in the ensemble using a randomly drawn subset of the training set [32].

[7]AdaBoost: is an ensemble algorithm that attempts to produce a very accurate classification rule by combining moderately inaccurate weak classifiers [35].

[8]Functional Tree: are classification trees that could have logistic regression function at the inner nodes and or leaves [35].

[9]Random Tree: is an ensemble classifier that consists of directed graphs build with a random process [35].

[10]RBF Network is a neural network that consists of input nodes connected by weights to a set of RBF neuron, which fire proportionally to the distance between the input and the neuron in weight space. A Radial Basis Function (RBF) is a real-valued function whose value depends only on the distance from the origin. [11].

[11]RIPPER [48] is a rule-based classifier that builds a set of rules that identify the classes while minimizing the amount of error. The error is defined by the number of training examples misclassified by the regulations.

[12]Zero-R [9] is the simplest classifier which always predicts the majority class in training set. It is commonly employed for determining a baseline performance as a benchmark for other methods.

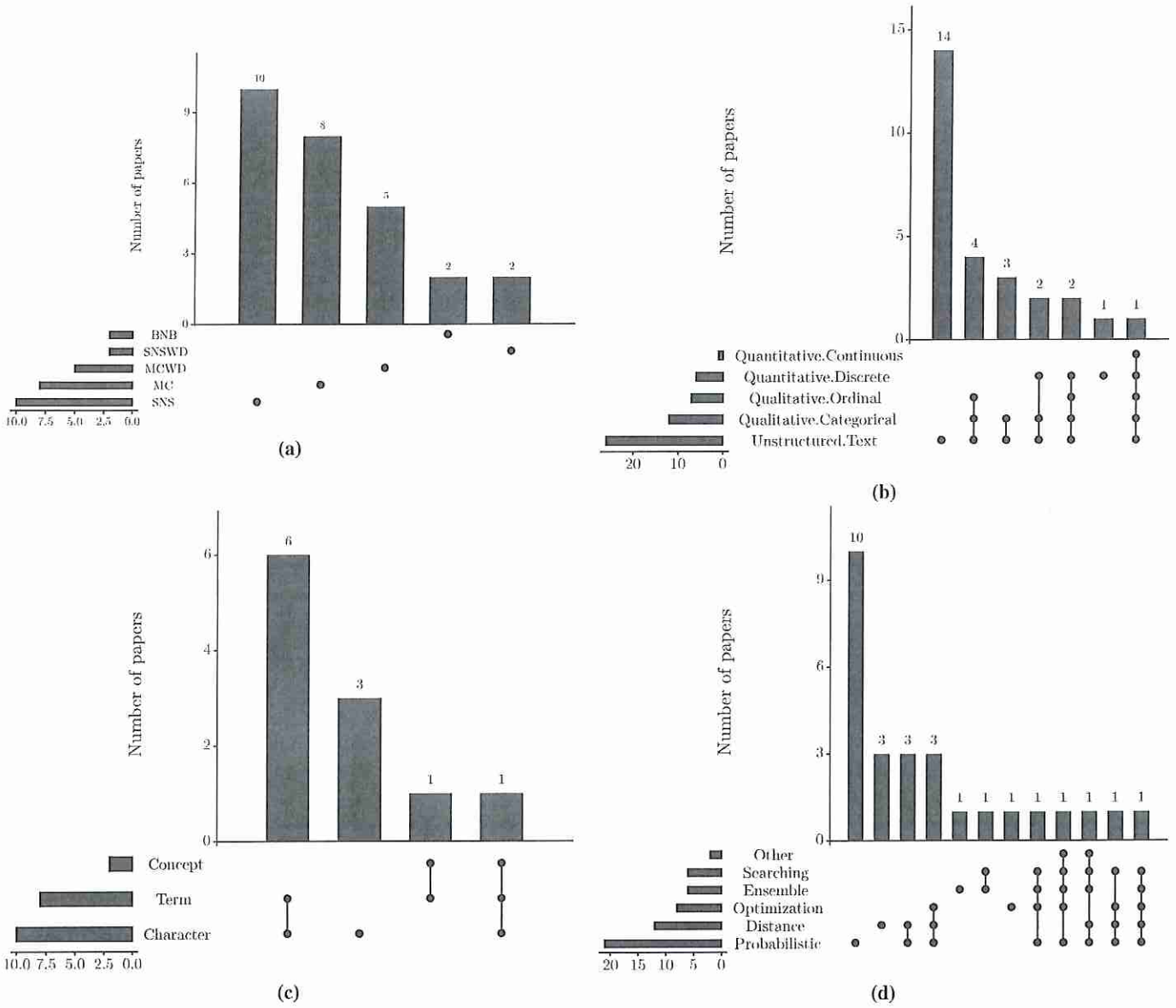

Fig. 6: (a) Paper distribution by prediction problem category, (b) paper distribution by feature data category, (c) paper distribution by text mining method category (d) paper distribution by ML algorithm categories.

of the single class category (21 out of 27 ≈ 70%), as shown Figure 7a. Only one paper (Roy et al. [29]) investigated all four categories of measures presented in the chart.

I. Which sampling techniques are applied most frequently to generate more reliable predictive performance estimates in severity prediction of a bug report (RQ9)?

The most of the papers shown in Table XV applied 10-Fold CV (≈ 66%) to generate more reliable estimates. Figure 7b displays that the majority of those papers (14 out of 15 ≈ 93%) employed a simple resampling method. Also, only one paper (Otoom et al. [35]) investigated both non-resampling and simple resampling methods for bug report severity prediction.

[13] MRR (Mean Reciprocal Rank) [32] is the average of reciprocal ranks of results of a set of questions. A reciprocal rank of a question is the multiplicative inverse of the rank of the first correct answer [49].

[14] Effectiveness Ratio is cost-effectiveness measure which evaluates prediction performance given a cost limit.

[15] Krippendorff's Alpha Coefficient [37] is statistical measure of the agreement achieved when coding a set o units analysis of values of a variable.

[16] Stratified k-fold CV [17] is a k-fold CV variation which ensures that class distribution is respected in the training and testing sets created at every fold.

[17] SMOTE (Synthetic Minority Oversampling TEchnique) [50] is an sampling approach in which the minority class is over-sampled by creating "synthetic" examples rather than by over-sampling with replacement.

TABLE XIV: Papers distribution by evaluation measures.

Measure	Category	References	Total
Precision	Single Class	[4], [3], [25], [27], [9], [28], [29], [8], [31], [32], [33], [34], [36], [38], [39], [40], [41], [42], [43], [44], [45]	21
Recall	Single Class	[4], [3], [25], [27], [9], [28], [29], [8], [32], [34], [36], [38], [39], [40], [41], [42], [43], [44], [45]	19
F-measure	Single Class	[4], [3], [25], [27], [9], [28], [29], [8], [32], [36], [38], [39], [40], [41], [42], [43], [44], [45]	18
Accuracy	Multi-Class	[25], [27], [9], [28], [29], [30], [31], [33], [34], [35], [44]	11
AUC	Summary	[4], [5], [24], [26], [29]	5
ROC	Graphical	[4], [29]	2
MRR[8]	Single Class	[27], [38]	2
Effectiveness Ratio[9]	Single Class	[32]	1
Krippendorff's Alpha Reliability[10]	Single Class	[37]	1

TABLE XV: Paper distribution by sampling methods.

Method	Category	References	Total
10-Fold CV	Simple Re-sampling	[5], [24], [25], [26], [32], [34], [35], [41], [43], [45]	10
Stratified 10-Fold CV[11]	Simple Re-sampling	[9], [29], [44]	3
SMOTE[12]	Simple Re-sampling	[32], [40]	2
Hold-out	No Re-sampling	[35]	1
03-Fold CV	Simple Re-sampling	[40]	1
05-Fold CV	Simple Re-sampling	[31]	1

J. Which statistical tests were used to compare the performance between two or more ML algorithms for bug report severity prediction (RQ10)?

As shown in Table XVI, most of the papers used Wilcoxon Signed Rank Test (8 out of 27 - around 29%) and T-test (7 out of 27 - around 26%) to compare the performance of ML algorithms. Figure 7c calls attention that only ten papers (around 37%) reported using any statistical test in their experiments to bug report severity prediction. The figure also shows that the most of these papers (8 out of 10 - around 80%) applied a non-parametric test, and 5 out of 10 (50%) papers employed both parametric and non-parametric tests.

TABLE XVI: Paper distribution by statistical tests.

Test	Category	References	Total
Wilcoxon signed rank test	Non-parametric	[27], [9], [8], [32], [38], [41], [43], [44]	8
T-test	Parametric	[27], [8], [38], [40], [41], [43], [45]	7
Proportion test	Parametric	[45]	1
Shapiro-Wilk test	Parametric	[43]	1

K. Which software tools were used to run bug report severity prediction experiments (RQ11)?

Most of the papers listed in Table XVII used RapidMiner (5 out of 18 - around 27%) to run their experiments. However, Figure 7d demonstrates a strong predominance of FLOSS over CSS, 13 out of 18 papers (around 61%) executed FLOSS to perform their experiments. The figure also shows that no paper used both CSS and FLOSS.

L. Which solution types were proposed for bug report severity prediction problem (RQ12)?

Most of the papers (26 out of 27 - around 93%) provided offline solutions (Table XVIII). Just one paper (Tian et al. [3]) presented a online solution.

TABLE XVII: Paper distribution by software tools.

Tool	Category	References	Total
RapidMiner	CSS	[25], [28], [31], [33], [44]	5
WEKA	FLOSS	[5], [32], [34], [37]	4
R	FLOSS	[27], [8], [43]	3
NLTK	FLOSS	[29], [38], [41]	3
Statisca	CSS	[25], [28]	2
TMT	FLOSS	[27], [38]	2
OpenNLP	FLOSS	[3]	1
WordNet	FLOSS	[29]	1
Ruby	FLOSS	[4]	1
WVTool	FLOSS	[24]	1
CoreNLP	FLOSS	[43]	1

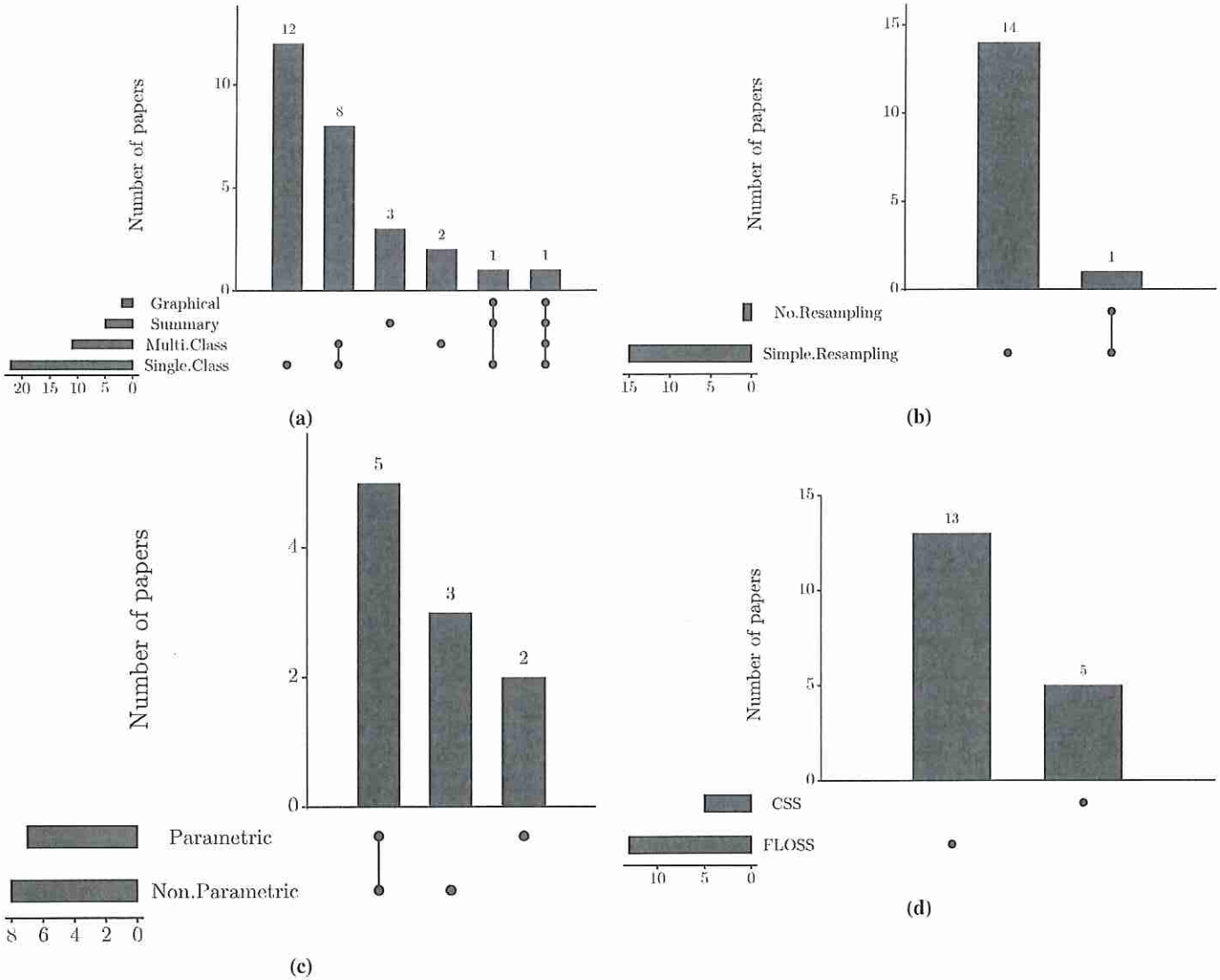

Fig. 7: (a) Paper distribution by evaluate measure categories, (b) Paper distribution by sampling method categories, (c) Paper distribution by statistical test categories, and (d) Paper distribution by software tool categories.

TABLE XVIII: Solution types proposed in selected papers.

Tool	References	Total
offline	[4], [5], [24], [25], [27], [26], [9], [28], [29], [30], [8], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45]	26
online	[3]	1

VI. Discussion

This section presents lessons learned from this systematic mapping study and recommends possible research directions for the bug severity prediction problem.

Researchers have concentrated their focus on few FLOSS, namely Eclipse and Mozilla, both hosted by Bugzilla. The main reason given by most researchers is benchmarking their results with Lamkanfi et al. [4], [5], which are often considered the state-of-the-art in this researching area. Notably, they have not consider relevant FLOSS projects like as Linux Kernel, Spark, MySQL, among others. Moreover, published research only marginally investigated two others bug trackers mentioned in reviewed papers: Jira and Google Issue Tracker. The first one tracks bugs for more than 80% Apache Foundation's projects[18], and the last one tracks bugs for many internal and external Google Inc. projects[19]. No paper investigated Github, an other outstanding collaborative website which hosted more than 67 million projects (including many FLOSS)[20].

From 14 FLOSS identified in this review, six are programming tools, four are system software, and four are application software. For two formers, people that reports bug are typically technical users, and, for the latter, people are end-users. A bug report can be more or less detailed depending who reports it [4] [5]. Therefore, the probability of writing more accurate bug reports is larger for technical users than end-users.

The bug report severity prediction is a single problem, but researchers addressed it considering five kinds of the classification problem. Two of them not considered default severity level (often "normal") as a valid class. By doing so, they have argued that they would avoid noisy and unreliable data. Saha et al. [30] call attention that many bug reports in practice are not "normal" and this misclassification can affect the accuracy of ML algorithms. However, the same authors do not recommend just discarding these bug reports ~~from dataset~~.

Most of the approaches proposed by researchers to predict bug report severity relied on unstructured text features (summary and description). Because of this, text mining techniques played, side by side with machine learning methods, an essential role in delivering appropriated solutions. These approaches quite used two others (product and component), both qualitative. That may indicate that bug reports collected from separate parts of a single FLOSS yield different ML algorithms outcomes.

Regarding available feature data types in bug reports, more than half of them are qualitative. SVM and Neural Networks, two popular and essential ML algorithms in modern machine learning [11], do not work with qualitative data [10]. These two facts bring an extra challenge to use qualitative features for bug report severity prediction. When input data set has qualitative data, categorical and ordinal values must be converted into numeric values before applying ML algorithms.

Despite a large number of features raised by the application of text mining activities (tokenizing, stop word removal and stemming) on summary and description, few papers employed feature selection methods for bug severity prediction. Furthermore, these papers notably only investigated filter feature selection methods. Filter approach has two drawbacks [10]: (i) it does not take into account redundancy between features, and (ii) it does not detect dependencies between them.

Most papers did not explicitly report using of text ming methods. Few of them reported to use mainly TF-IDF (for feature vector weighting), unigram (for feature extraction) and BM25 (for verifying text similarity).

It seems that researchers have adopted a conservative posture regarding the use of ML algorithms. Most papers applied at least one of the following well-known and traditional supervised algorithms: k-NN, Näive Bayes and its extension Näive Bayes Multinomial.

Researchers evaluated the performance of ML algorithms in their proposed approaches using precision, recall, and f-Measure. Three common measures employed in ML arena, but which may strongly skew in the imbalanced scenery. This characteristic is particularly critical in bug report repositories which are intrinsically imbalance in practice. To minimize such distortion, Tian et al. [37], instead, recommend measuring performance using inter-rater agreement based metrics, such as Cohen's Kappa [51] or Krippendorff's Alpha [52]. Despite its public issues in an imbalanced dataset [17], quite number of papers still used accuracy to measure ML performance. AUC, an alternative used by few number of papers, seems more robust than accuracy in imbalanced data conditions. Like Kappa and Krippendorff's Alpha, it considers class distribution on performance evaluation of ML algorithms for bug report severity prediction.

Few papers reported using a resampling approach to improving the accuracy of ML algorithms. All methods reported ~~belongs to~~ simple resampling ~~category~~ strategies, including, the most used, k-fold cross-validation. Only one paper used SMOTE, considered

[18] http://www.apache.org/index.html#projects-list, as of July 3, 2018.
[19] https://developers.google.com/issue-tracker/, as of July 3, 2018.
[20] https://octoverse.github.com/, as of July 3, 2018.

a "de facto" resampling method in imbalanced data scenario [53]. Japkowicz et el. [17] also suggests that experiments may achieve better results using multiple resampling methods, for example, repeated k-fold cross-validation.

It seems that most researchers compared the results yielded in their experiments with others observing only the means of measures applied. Few of them reported utilizing statistical tests to do that, which is a recommended practice in empirical software engineering [54]. Most used statistical test was parametric. This kind of test require that samples follow a statistical distribution (e.g., normal), which is not guaranteed to occur in practice when comparing ML models [18]. Besides, No paper investigated Analysis of Variance (ANOVA), a robust ranked-based test recommended by Kitchenham et al. [54] for Empirical Sofware Engineering.

Researchers preferred to use FLOSS tools to perform their analyses. Although, a reasonable number of papers have used a proprietary software named RapidMiner. It calls attention that researchers misused top-ranked ML tools based on R and Python programming languages.

It seems that most of the proposed solutions for bug report severity prediction were conceived to run in offline mode, just one paper claimed which the published solution is mature to work in a production setting. The need for a high number of labeled bug reports during the training phase [55] is a problem intrinsic to supervised ML algorithms may make it difficult to turn these solutions to online.

VII. Conclusions and Research Directions

A systematic mapping review provides a structure for a type of research reports, categorizing and giving a visual summary of results that have been published in papers of a research area [7]. This map aids to identify gaps in a research area, becoming a basis to guide new research activities. [6]. The current mapping review captured the current state of research on bug report severity prediction, characterized related problems and identified the main approaches employed to solve them. These objectives were reached by conducting mapping of existing literature. In total, the review identified 27 relevant papers and analyzed them along 12 dimensions. Although these papers have made valuable contributions in bug report severity prediction, the panorama presented in this mapping review suggests that there are potential research opportunities for further advancement in this topic and to produce better results. Among them, the following research directions appear to be more promising:

- There is an apparent lack of investigation on bug report severity prediction in others relevant FLOSS such as, for example, Linux Kernel, Ubuntu Linux, and MySQL, and in others BTS, for example, Github, to confirm the published results.
- The users who wrote the most bug reports investigated in reviewed papers were technicals users. Thus, the influence of user experience in prediction outcomes was underinvestigated.
- Bug reports labeled with default severity level (often "normal") were prevalent in the most datasets used in reviewed papers. However, they are considered unreliable [30], and just discarding them also does not seem appropriate. Then, efforts in researching on novel approaches to treat this type of report should be considered to improve the state-of-the-art of severity prediction algorithms.
- Most approaches published in reviewed papers were based on unstructured texts features (summary and description). To treat them, researchers chose to use traditional bag-of-words instead of more recent text mining methods (e.g., word-embedding) which may likely to improve outcomes yielded so far.
- There is a clear research opportunity to investigate whether state-of-the-art ML algorithms might outperform the traditional algorithms used in all reviewed papers for bug report severity prediction. Deep learning algorithms which perform very well when classifying audio, text, and image data [56] could be a possible research direction.
- Researchers should investigate more recent techniques (e.g., continuous learning) to provide a approach for bug report prediction which could be employed in real conditions.
- One suspects that many bug reports are resolve in few days (or in few hours). Efforts to predict severity level for these group of bug reports not seem very useful. Thus, an investigation to confirm this hypothesis and to determine when the severity prediction is more appropriate in bug report lifecycle is critical.

Acknowledgment

Authors are grateful to CAPES (grant #88881.145912/2017-01), CNPq (grant #307560/2016-3), FAPESP (grants #2014/12236-1, #2015/24494-8, #2016/50250-1, and #2017/20945-0) and the FAPESP-Microsoft Virtual Institute (grants #2013/50155-0, #2013/50169-1, and #2014/50715-9).

References

[1] Y. C. Cavalcanti, P. A. da Mota Silveira Neto, I. d. C. Machado, T. F. Vale, E. S. de Almeida, and S. R. d. L. Meira, "Challenges and opportunities for software change request repositories: a systematic mapping study," *Journal of Software: Evolution and Process*, vol. 26, no. 7, pp. 620–653, jul 2014.

[2] I. Sommerville, *Software Engineering*, 2010.

[3] 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*, Oct 2012, pp. 215–224.

[4] 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)*, May 2010, pp. 1–10.

[5] 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*, March 2011, pp. 249–258.

[6] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," 2007.

[7] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE'08. Swindon, UK: BCS Learning & Development Ltd., 2008, pp. 68–77.

[8] T. Zhang, G. Yang, B. Lee, and A. T. S. Chan, "Predicting severity of bug report by mining bug repository with concept profile," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ser. SAC '15. New York, NY, USA: ACM, 2015, pp. 1553–1558.

[9] H. Valdivia Garcia and E. Shihab, "Characterizing and predicting blocking bugs in open source projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 72–81.

[10] P. Flach, *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. New York, NY, USA: Cambridge University Press, 2012.

[11] S. Marsland, *Machine Learning: An Algorithmic Perspective, Second Edition*, 2nd ed. Chapman & Hall/CRC, 2014.

[12] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[13] G. Williams, *Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery*, 2011.

[14] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[15] Z. Zheng, X. Wu, and R. Srihari, "Feature selection for text categorization on imbalanced data," *SIGKDD Explor. Newsl.*, vol. 6, no. 1, pp. 80–89, Jun. 2004.

[16] R. Feldman and J. Sanger, *Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. New York, NY, USA: Cambridge University Press, 2006.

[17] N. Japkowicz and M. Shah, *Evaluating Learning Algorithms: A Classification Perspective*. New York, NY, USA: Cambridge University Press, 2011.

[18] K. Facelli, A. C. Lorena, J. Gama, and A. C. d. Carvalho, *Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina*. Rio de Janeiro, RJ, Brasil: LTC, 2015.

[19] F. Wilcoxon, *Individual Comparisons by Ranking Methods*, S. Kotz and N. L. Johnson, Eds. New York, NY: Springer New York, 1992.

[20] N. Lewis, *100 Statistical Tests In R*, ser. Easy R Series. Createspace Independent Pub, 2013.

[21] J. Uddin, R. Ghazali, M. M. Deris, R. Naseem, and H. Shah, "A survey on bug prioritization," *Artificial Intelligence Review*, vol. 47, no. 2, pp. 145–180, Feb 2017.

[22] M. Brhel, H. Meth, A. Maedche, and K. Werder, "Exploring principles of user-centered agile software development: A literature review," *Information and Software Technology*, vol. 61, pp. 163 – 181, 2015.

[23] Érica Ferreira de Souza, R. de Almeida Falbo, and N. L. Vijaykumar, "Knowledge management initiatives in software testing: A mapping study," *Information and Software Technology*, vol. 57, pp. 378 – 391, 2015.

[24] C. Z. Yang, C. C. Hou, W. C. Kao, and I. X. Chen, "An empirical study on improving severity prediction of defect reports using feature selection," in *2012 19th Asia-Pacific Software Engineering Conference*, vol. 1, Dec 2012, pp. 240–249.

[25] 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, no. 2, pp. 32–59, 2012.

[26] 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*, June 2014, pp. 216–219.

[27] 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*, July 2014, pp. 97–106.

[28] M. Sharma, M. Kumari, R. K. Singh, and V. B. Singh, "Multiattribute based machine learning models for severity prediction in cross project context," in *Computational Science and Its Applications – ICCSA 2014*, B. Murgante, S. Misra, A. M. A. C. Rocha, C. Torre, J. G. Rocha, M. I. Falcão, D. Taniar, B. O. Apduhan, and O. Gervasi, Eds. Cham: Springer International Publishing, 2014, pp. 227–241.

[29] 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*, Aug 2014, pp. 269–276.

[30] R. K. Saha, J. Lawall, S. Khurshid, and D. E. Perry, "Are these bugs really "normal"?" in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, May 2015, pp. 258–268.

[31] 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, proceedings of the 4th International Conference on Eco-friendly Computing and Communication Systems.

[32] X. Xia, D. Lo, E. Shihab, X. Wang, and X. Yang, "Elblocker: Predicting blocking bugs with ensemble imbalance learning," *Information and Software Technology*, vol. 61, pp. 93 – 106, 2015.

[33] S. Gujral, G. Sharma, S. Sharma, and Diksha, "Classifying bug severity using dictionary based approach," in *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, Feb 2015, pp. 599–602.

[34] M. N. Pushpalatha and M. Mrunalini, "Predicting the severity of bug reports using classification algorithms," in *2016 International Conference on Circuits, Controls, Communications and Computing (I4C)*, Oct 2016, pp. 1–4.

[35] 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)*, April 2016, pp. 92–95.

[36] K. K. Sabor, M. Hamdaqa, and A. Hamou-Lhadj, "Automatic prediction of the severity of bugs using stack traces," in *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering*, ser. CASCON '16. Riverton, NJ, USA: IBM Corp., 2016, pp. 96–105.

[37] Y. Tian, N. Ali, D. Lo, and A. E. Hassan, "On the unreliability of bug severity data," *Empirical Softw. Engg.*, vol. 21, no. 6, pp. 2298–2323, dec 2016.

[38] 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.

[39] K. Jin, A. Dashbalbar, G. Yang, J.-W. Lee, and B. Lee, "Bug severity prediction by classifying normal bugs with text and meta-field information," *Advanced Science and Technology Letters*, vol. 129, pp. 19–24, 2016.

[40] T. Choeikiwong and P. Vateekul, "Improve accuracy of defect severity categorization using semi-supervised approach on imbalanced data sets," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, 2016.

[41] K. Jin, A. Dashbalbar, G. Yang, and B. Lee, "Improving predictions about bug severity by utilizing bugs classified as normal," *Contemporary Engineering Science*, vol. 9, pp. 933–942, 2016.

[42] K. Jin, E. C. Lee, A. Dashbalbar, J. Lee, and B. Lee, "Utilizing feature based classification and textual information of bug reports for severity prediction," *Information*, vol. 19, no. 2, pp. 651–659, Feb 2016.

[43] 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*, ser. SAC '17. New York, NY, USA: ACM, 2017, pp. 1280–1287.

[44] V. B. Singh, S. Misra, and M. Sharma, "Bug severity assessment in cross project context and identifying training candidates," *Journal of Information and Knowledge Management*, vol. 16, no. 01, p. 1750005, 2017.

[45] N. K. S. Roy and B. Rossi, "Cost-sensitive strategies for data imbalance in bug severity classification: Experimental results," in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Aug 2017, pp. 426–429.

[46] R. Pressman, *Software Engineering: A Practitioner's Approach*, 7th ed. New York, NY, USA: McGraw-Hill, Inc., 2010.

[47] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, Mar. 2003.

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

[49] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? - more accurate information retrieval-based bug localization based on bug reports," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 14–24.
[50] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *J. Artif. Int. Res.*, vol. 16, no. 1, pp. 321–357, Jun. 2002.
[51] A. Ben-David, "Comparison of classification accuracy using cohen's weighted kappa," *Expert Systems with Applications*, vol. 34, no. 2, pp. 825 – 832, 2008.
[52] K. Krippendorff, "Computing krippendorff's alpha-reliability," *Departamental Papers (ASC)*.
[53] A. Fernandez, S. Garcia, F. Herrera, and N. V. Chawla, "Smote for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary," *Journal of Artificial Intelligence Research*, vol. 61, pp. 863–905, Apr. 2018.
[54] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong, "Robust statistical methods for empirical software engineering," *Empirical Software Engineering*, vol. 22, no. 2, pp. 579–630, Apr 2017.
[55] A. Nigam, B. Nigam, C. Bhaisare, and N. Arya, "Classifying the bugs using multi-class semi supervised support vector machine," in *International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME-2012)*, March 2012, pp. 393–397.
[56] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.