Bug Report Severity Level Prediction in Free Libre/Open Source Software: A State of Art Survey

Author1 Affiliation1 Author2 Affiliation2 Author2 Affiliation2

Abstract

The bug report severity level is considered a important variable for planning software maintenance activities in bug tracking systems. It indicates how soon a bug needs to be addressed. However, the severity level assignment primarily remains a manual process, mostly depending on the experience and expertise who has reported or reviewed the bug report.In FLOSS, because of its peculiarities, this problem becomes more critical generating a much greater impact over the maintenance process. Due to the importance of this topic, both business and academic community, there has been a great interest in this topic, yielding ~~to~~ an extensive research effortto provide computational solutions to predict automatically the bug report severity. 27 papers published in 23 computer science journals and conferences from 2010 to 2017, shown that machine learning and text mining algorithms have played a central role in this area. This systematic mapping study quantitatively characterized many aspects of the experiments performed in these papers: FLOSS investigated, prediction problem kind, features and selection methods, text mining methods, machine learning algorithms, performance evaluation measures, sampling techniques, statistical tests, experiment tools, and type of solution delivered. The gathered data reflects the relevance of bug report severity prediction to maintenance activities in FLOSS projects, the scientific maturity of this research topic, as well as, to identifies gaps which can motivate new research initiatives .

Index Terms

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

I. Introduction

Bug tracking systems (BTS) have played a significant role in maintenance process in many software development settings. In Free/Libre Open Source Software (FLOSS) scenario, this is especially true. Since, FLOSS projects are characterized by the existence of many of 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 him to communicate a bug to who developed or is responsible for maintaining a software [2]. Initially, he or she should inform a short description, a long description, a type (e.g., bug, new feature, enhancement, and task) 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, this team member can complete the information in bug report form, indicating, for example, its priority and assigning the person responsible for fixing the bug.

The severity level information is recognized as a critical variable in the equation to estimate a prioritization of bug reports [3]. It 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]. Consequently, it is a process with a high degree of subjectivity, and it may be quite error-prone.

The number of bug reports in large and medium software FLOSS projects is frequently very large [5]. Eclipse project received over 2,764 requests and GNOME project received over 3,263 requests between 01/10/2009 and 01/01/2010. Severity level shifts throughout CR lifecycle may disturb the planning of maintenance activities. For example, the maintenance team could be assigned to address less significant CRs before most important ones. There have been reports [6] of efforts to implement intelligent software assistants to help developers and maintenance personnel in defining more accurately the field values in a CR form. Currently, Machine Learning techniques have become a popular method to address this issue, and there are quite a few publications in this area in the literature [1].

Machine Learning (ML) techniques have been successfully applied in solving real problems in many areas of knowledge, including those related to BTS software, such as duplication and assignment of CR [1]. However, the accuracy of ML algorithms may be affected by imbalanced datasets [7] — a recurring critical problem in bug report repositories [6]. For example, more than 60% of bug reports may have a "major" severity level. In addition to this problem, most publications are still focused on predicting the severity level of bug reports, and none of them have been implemented into popular tools like as Bugzilla, Jira, and Redmine [1]. Furthermore, many have used proprietary and not public ML algorithms. Therefore, there is still a clear need of advances in this knowledge area, especially broadening the reach of research questions and including more popular and open OSS and ML algorithms.

Due to the relevance of bug report severity information and the potential benefits of its correct prediction for the planning of FLOSS maintenance, the current mapping study aims to identify state of the art on the prediction of bug report severity. This kind of study provides a broad overview of a research area to determine whether there is research evidence on a particular

topic. According to Kitchenham et al. [8] and Petersen et al. [9], results yielded of a mapping study help recognizing gaps to suggest future research and provide a direction to position new research activities appropriately.

There are some literature reviews related to bug report. Although existing works aided to comprehend 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 review investigates 27 papers — published from 2010 to 2017 — related to bug report severity prediction, analyzing them from more ten different perspectives. In this sense, this paper contributes to improving the knowledge about the research topic making available an in-depth study which identified and summarized the main topics researched in the papers.

The paper is organized as follows. Section III presents related work that are relevant to our research. Section II provides the information background about CR systems, text mining and machine learning techniques necessary to understand the research area. Section V presents our results. Section VI presents final findings and discussion. Finally, Section VII present conclusions and future work.

II. BACKGROUND

This section comments on basic concepts necessary to understand this research area, namely bug tracking systems, machine learning Learning [10], and text mining [11]. Two techniques frequently used for bug report severity prediction.

A. Bug Tracking System

Bug Tracking Systems (BTS) [4] is a software employed to keep the recording and tracking information of requests for modifications, bug fixes, and support that could occur during the software life cycle. While reporting a bug, a user is asked to provide information about the bug by filling out a form.

Although there is no a common sense regarding the terminology or the amount of information that users must fill in to complete his requisition, between popular BTS (e.g., Bugzilla, Jira, and Redmine) [3], typically, they shall fill in a form containing at least the following attributes shown in Table I.

TABLE I: Commom attributes in the BTS form.

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

Once the user has registered the report, the development team will assess it and, if it not canceled for some reason (e.g., duplication), they will complement the information with, for example, the person responsible for handling this request or a new severity level for the report. All these data are stored in a repository, keeping relevant historical data about a particular software.

The bug report status shifts throughout its lifecycle. Figure 1 shows a typical set of states that a bug report status attribute can hold. Each transition occurs because of a specific event as shown in the diagram.

During the bug report life cycle, the severity level can change in some states. Figure 2 explains when a user or a developer should assign a severity level to a bug report, according to standard Bugzilla severity levels. Researchers sometimes groups these levels in severe (blocker, critical, and major) and non-severe (normal, minor and trivial) to work with a coarse-grained classification problem. Also, some of them ignore "normal," because they consider this level as a gray region. At other times, researchers attempt to predict a bug report severity as blocking or non-blocking. A blocking bug is that prevents others bugs to be fixed [13].

B. Machine Learning

Machine Learning (ML) [14] 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 kinds 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, and regression when the label value is continuous.

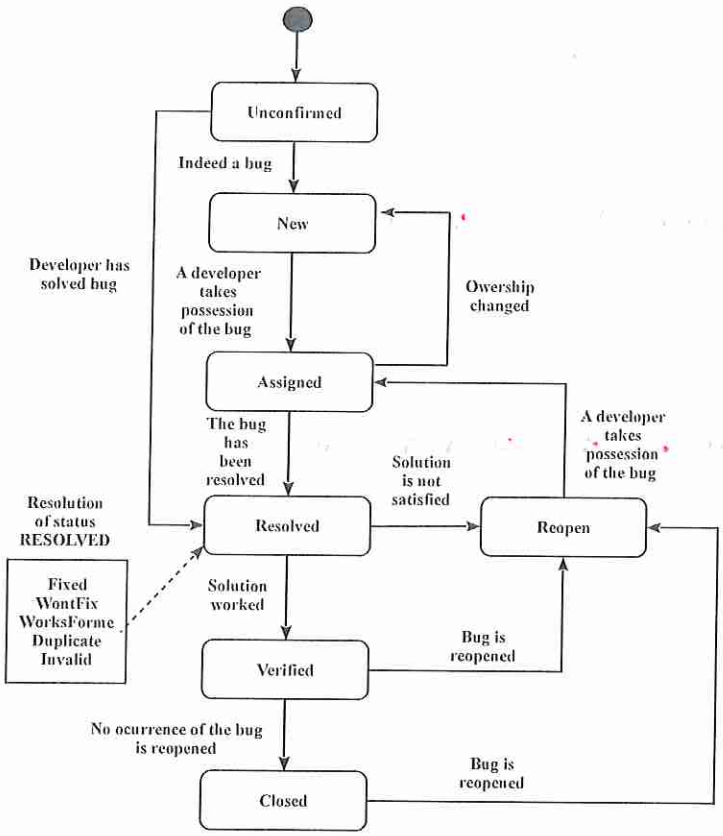


Fig. 1: Bug report life cycle [12].

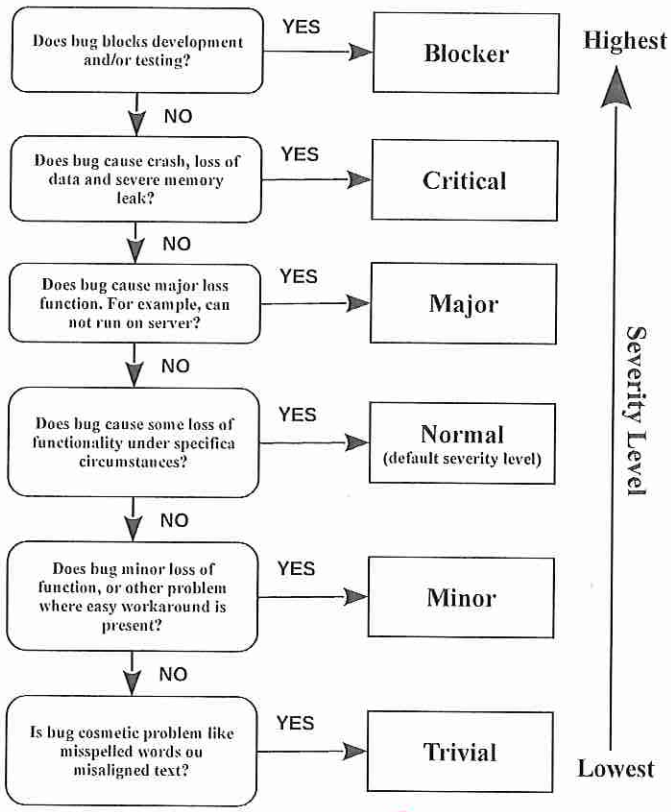


Fig. 2: Standard Bugzilla severity levels(https://www.bugzilla.org)

As of June 2018

Data clustering

A descriptive algorithm explores or describes a dataset. Differently, from the predictive algorithms, there is not an output label associated with a sample. ~~Grouping data~~ and pattern discovery are two examples of descriptive tasks. More detailing of descriptive algorithms are outside the scope of this paper.

A ML algorithm works over um dataset which contains many samples or instances x_i, where $i = \{1..n\}$. Each instance is composed ~~by~~ of $\{x_{i1}, x_{i2}, ..., x_{id}\}$ input attributes or independent variables, where $d = \{1..n\}$, and one output attribute or dependent variable, $x_{i(n+1)}$. Input attributes are commonly named features and output attribute as commonly named class attribute.

classification scenario

The most traditional ML algorithms are k-Nearest Neighbors, Naive 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 ~~scenery~~. Following a brief description of each one:

Provide reference

- **k-Nearest Neighbors** (k-NN) process the available instances (or neighbors) in a dataset and classifies a new instance based on its similarity ~~measure.~~ Usually, k-NN algorithms utilize a 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.

  to the k nearest neighbors

- **Naïve 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 $P(C_k|x)$. Naïve Bayes classifiers assume that given the class variable the value of a particular feature is independent of the value of any other feature.

  padronizar

- **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 [15]. It structures its processing as an interconnected group of artificial neurons. Figure 3 presents an artificial neuron model [16] 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.

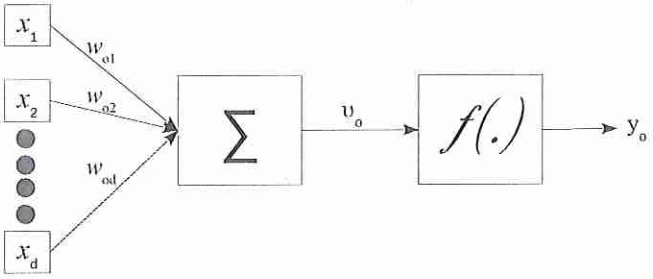


Fig. 3: A artificial neuron model based on Marsland [16].

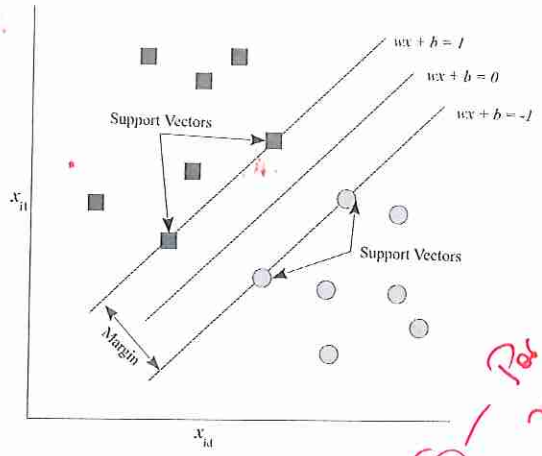
- **Support Vector Machine(SVM)** represents each instance as points in n-dimensional space. The algorithm divides every point of the separate categories by a clear gap that is as wide as possible [16]. This space between these two regions delimits the margin between classes. In Figure 4, for example, SVM will search for support vectors which are instances that found lie at the edge of an area in space, which presents a boundary between one of these classes instances (e.g., the squares) and another class observations (e.g., the circles). Each region contains observations with the same value for the target variable (i.e., the class) [17]. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.
- **Random Forest** [18] relies on two core principles: (i) ~~in~~ the creation of hundreds of decision trees and ~~then joining them~~ their combination into a single model; and (ii) in the closing decision based on the ruling of the majority of the forming trees which are treated as equals.

  a classification method

From Information Retrieve(IR) ~~discipline~~ core, the three most common performance measures for evaluating the accuracy of classification algorithms are precision, recall, and F-measure [11]:

- **Recall** for a class 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. More formally [19]: the recall is the number of True Positives(TP) divided by the number of True Positives(TP) and the number of False Negatives(FN), where the TP and FN values are derived from the confusion matrix. A low recall indicates many False Negatives [20].
- **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. More formally [19]: the precision is the number of True Positives(TP) divided by the number of True Positives and False Positives(FP), as well as TP and FN, FP also comes from the confusion matrix. A low precision can also indicate a large number of False Positives [20].

Present the equation



Por que você não cria a sua figura?

Fig. 4: A SVM model example [17].

times

- **F-measure** conveys the balance between the precision and the recall and combines the two measures in an ad hoc way [11], [20].F-measure can be calculated using the formula $2 * ((precision * recall))/(precision + recall)$.

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 [20]. 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: [19]:

- **Holdout** splits a dataset into a ratio of p for training and $(1 - p)$ for testing.
- **Cross-Validation** 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.

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 these outcomes may be difficult and not recommended [19]. It is necessary to conduct statistical ~~testing~~ tests for reliable performance comparison between models under investigation. The primary purpose of statistical testing is to gather evidence about which results yielded by an evaluation measures represents a general behavior of classifiers [20]. Statistical tests can be applied in two situations: comparing the performance of the models in a single dataset (e.g., Wilcoxon's Signed Rank), ~~and~~ or considering several datasets (e.g., Friedman's Test).

C. Text Mining

points of an n-dimensional space

of

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. Text mining is the process to convert unstructured text into a structure suited to analysis [11]. It is composed of three primary activities [17]: tokenization, stop word removal and stemming:

- **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).

approach

relies on the use of

In this representation

One of the most traditional ways of representing a document, ~~commonly known as~~ a bag of words [11]. ~~It uses~~ all ~~words~~ ~~in a text as the~~ features, and thus the dimension of the feature space is equal to the number of different ~~words~~ in all texts. ~~The~~ methods ~~of giving~~ weights ~~to the~~ 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 numbers 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 [11].

~~rep on represent~~

~~for assigning~~

terms

III. RELATED WORK

~~It was find~~ Two studies that review 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 papers 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. ~~Even though~~ However only two papers (Lamkanfi et al. [4] and Lamkanfi et al. [5]) addressed severity prediction on FLOSS projects. This mapping study 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 us 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] surveyed published articles 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 categorize 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 study 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 related to bug report severity prediction. Furthermore, this review considers ~~many~~ 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 study for identifying and analyzing relevant papers. It ~~based~~ on the software engineering systematic literature review guidelines and recommendations proposed by Kitchenham et al. [8], Brhel et al. [22], Souza et al. [23], and Petersen et al. [9]. 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 study is to provide a current status view of the research on bug report severity prediction in FLOSS. To ensure an unbiased selection process, this review addresses the following research questions.

- **RQ$_1$. When and where have the studies been published?** This research question gives to the researcher a perception whether the topic of this mapping study seems to be broad and new, providing an overview about where and when papers were published.
- **RQ$_2$. 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 bug report severity prediction problems. 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.
- **RQ$_3$. 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. Comprehending each solution provided for each prediction problem type is essential.
- **RQ$_4$. What are the most common feature used 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.
- **RQ$_5$. 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.
- **RQ$_6$. 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.
- **RQ$_7$. 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 teach of other algorithms to improve state of the art.

RQ_8. 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.

RQ_9. Which sampling techniques were employed to improve ML algorithms performance for bug report severity prediction? 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.

RQ_{10}. What statistical tests were used to compare the performance between two or more ML algorithms for bug report severity prediction? The answer of this research question can be useful to identify which statistical tests were used in bug report severity prediction context. In addition, it allows to grasp existing protocols for the comparison between two or more ML algorithms performance using statistical significance tests.

RQ_{11}. What software tools were used to run bug report severity prediction experiments? 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.

RQ_{12}. What solution types were proposed for bug report severity prediction problem? This research question examines whether the most common solutions proposed in the selected studies are either online or off-line. This 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).

NO

B. Study selection

data storage procedures

This section describes the selection process carried out through the mapping study. It comprises four steps: (i) terms and searching string; (ii) sources for searching; (iii) inclusion and exclusion criteria; and (iv) ~~how to store data.~~

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 mapping study.

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 mapping study, it was selected four electronic databases and one popular search engines recommended by Kitchenham et al. [8]. Table III shows each selected 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 study 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 study does not have an abstract;

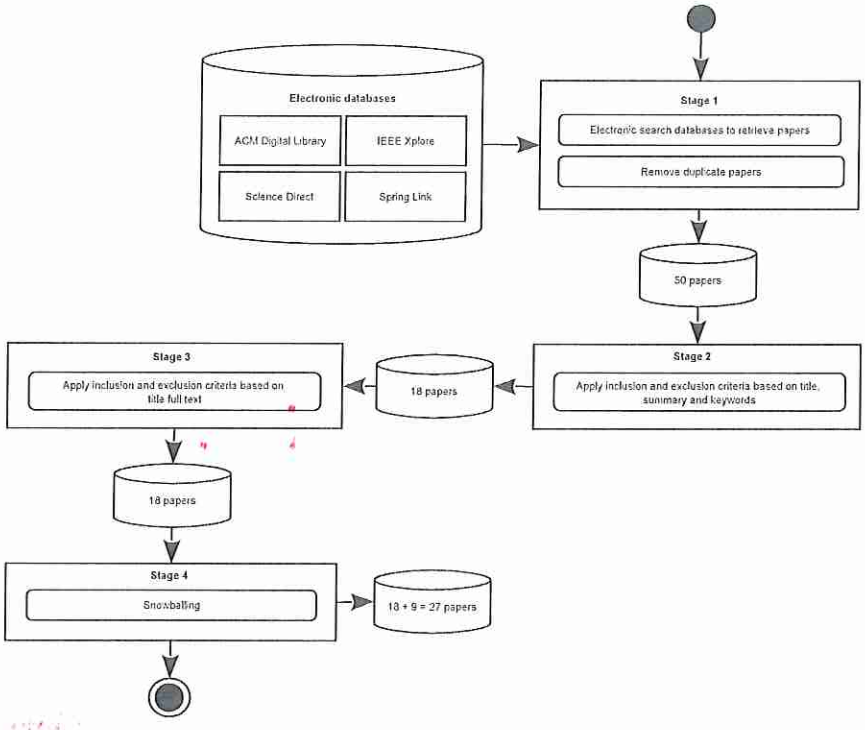


Fig. 5: Diagram of the four-stage paper selection process.

EC$_2$. The study is just published as an abstract;
EC$_3$. The study is written in a language other than English;
EC$_4$. The study is not a primary study (e.g., keynotes);
EC$_5$. The study 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 mapping study.

5) Assessment: According to Kitchenham et al. [8], the consistency of the protocol utilized in a mapping study 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 review process was conducted by the second and third authors.

C. Data extraction and synthesis

Figure 5 illustrates the four-stage selection process carried out in the current mapping study. 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 study discusses severity prediction of bug reports in FLOSS projects). In the third stage, exclusion criteria were applied considering the full text. However, no study was rejected by taking into account these criteria.

In the fourth stage, the Snowballing [8] 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 study is written in a language other than English); EC$_6$ eliminated one more paper (the study is inaccessible 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 identifier (#id) for each paper. Throughout the remainder of this text, these identifiers will be used to refer to the corresponding paper.

TABLE IV: Selected studies

#ID	Bibliographic reference
#1	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.
#2	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.
#4	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.
#5	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.
#6	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.
#7	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.
#8	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.
#9	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.
#10	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.
#11	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.
#12	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.
#13	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.
#14	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.
#15	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.
#16	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.
#17	A. F. Otoom, D. Al-Shdaifat, M. Hammad, and E. E. Abdallah, "Severity prediction of software bugs," in *2016 7th International Conference on Information and Communication Systems (ICICS)*, April 2016, pp. 92–95.
#18	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.
#19	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.
#20	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.
#21	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.
#22	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.
#23	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.
#24	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.
#25	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.
#26	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.
#27	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.

D. Categorization scheme

Petersen et al. [9] suggest the definition of a categorization scheme for conducting a systematic mapping analysis. The categories defined for this mapping study 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 study.

1) FLOSS software type (RQ_2): This scheme organizes FLOSS projects by software type. Based on the taxonomy suggested by Pressman [45], 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.

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

a. **SNS**: This category comprises prediction problems whose severity of bug report can be predicted as either severe or non-severe. The default severity level is considered neither into severe or non-severe classes.
b. **SNSWD**: This category comprises prediction problems whose severity of bug report can be predicted as severe or non-severe. The default severity level is considered into severe or non-severe classes, or as a separate class.
c. **MC**: This category comprises prediction problems whose the severity of bug report can be predicted as one out of multiple classes. The default severity level is not considered as a valid class.
d. **MCWD**: This category comprises prediction problems whose the severity of bug report can be predicted as one out of multiple classes. The default severity level is considered as a valid class.
e. **BNB**: The prediction problem whose ML algorithms 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 [13].

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 with no natural sense of order. For instance, bug report platform attribute ("Windows", "Linux", "Android"). e.g.,
b. **Qualitative ordinal**: ~~This category encompasses~~ features with a natural sense of order. For instance, bug report priority attribute (P1, P2, P3).
c. **Quantitative discrete**: This category encompasses features that contain a finite or an infinite number of values that can be counted, ~~For instance, bug report~~ 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, ~~For instance~~ such as the attribute, bug report 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 usually includes ...

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 [46]:

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.
b. **Wrapper**: This category comprises methods that prepares, evaluates and compares feature combinations, and selects the best one. It considers the feature selection as a search problem.
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 text mining feature representations techniques used for bug report severity prediction in four categories. The following categories were employed in this mapping study according to [11]:

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 types (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. [19]:

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 classifiers to make a prediction.

results of different base

7) Evaluation measures categories (RQ_8): This scheme organizes evaluation measures used in selected papers by categories. According to Japkowicz and Shah [20], 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 [20] 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 the statistical tests used for bug report severity prediction in three categories. Such categories employed in this mapping study according to Japkowicz and Shah [20] 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, it is needed to consider the following common limitations of all mapping studies:

- **Mapping review completeness can never be guaranteed [22]**: Although in this mapping review, it has been observed a strict research protocol to ensure the relatively whole population of the relevant literature, some important papers might be missed.
- **Terminological 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 the mapping study. The answers to each research question (RQ1 to RQ12) are presented in tables, where every paper uses an id (#id) as its reference. The organization of extracted data followed the criteria defined in Section IV.

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

Figure 6a shows the temporal distribution of papers. Just over 75% (22/27) of primarily selected papers were published after 2012. Table V presents sources of the selected papers, their types, and their identifiers. Although four studies were published in the MSR and two in the SEAA conferences, it is worth observing that 21 out of 27 selected papers were published in 21 different venues. Figure 6b shows that conferences were preferred rooms to publish 67% of the selected papers, while journals were chosen by 33% of the papers.

TABLE V: Papers publication sources.

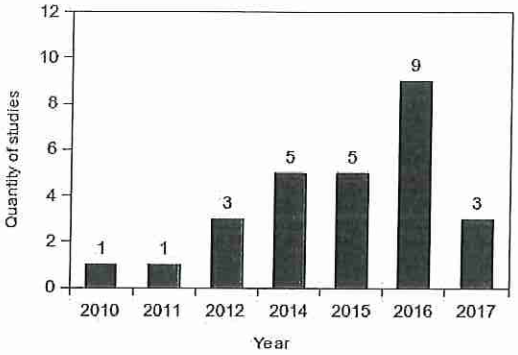
Paper source	Publication kind	#ID
ACM Symposium on Applied Computing	Conference	#12
Advanced Science and Technology Letters & Journal	Journal	#21
Asia-Pacific Software Engineering Conference	Conference	#4
Computational Science and Its Applications (ICCSA)	Conference	#9
Computer Software and Applications Conference	Conference	#6
Contemporary Engineering Sciences	Journal	#23
Empirical Software Engineering	Journal	#19
Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)	Conference	#15
Information and Software Technology	Journal	#14
International Conference on Circuits, Controls, Communications and Computing (I4C)	Conference	#16
International Conference on Computer Science and Software Engineering	Conference	#18
International Conference on Information and Communication Systems (ICICS)	Conference	#17
International Conference on Software Engineering and Service Science	Conference	#7
International Information Institute	Journal	#24
International Journal of Open Source Software and Process(IJOSSP)	Journal	#5
International MultiConference of Engineers and Computer Scientists	Conference	#22
Journal of Information & Knowledge Management	Journal	#26
Journal of Systems and Software	Journal	#20
Procedia Computer Science	Journal	#13
Software Engineering and Advanced Applications (SEAA)	Conference	#10, #27
Symposium on Applied Computing (SAC)	Conference	#25
Working Conference on Mining Software Repositories (MSR)	Conference	#1, #2, #8, #11
Working Conference on Reverse Engineering	Conference	#3

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

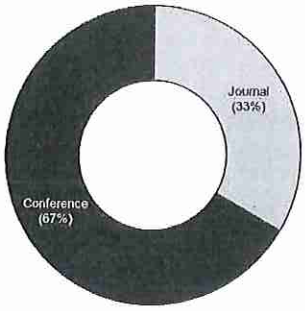
Table VI details the FLOSS found out after analysis of data gathered throughout this mapping study. The details for each FLOSS include a textual description; a category, according to Section IV-D; a BTS name, which hosted it; and an electronic address, from where it can be accessed. Table VII shows FLOSS distribution by selected papers. It can be observed that the top five projects were used for at least 11% of papers. Nearly 92% of papers employed Eclipse (25 out of 27), 70% Mozilla (19 out of 27), 18% Openoffice (5 out of 27), 14% Netbeans (4 out of 27) and 11% Gnome (3 out of 27).

The chart in Figure 7a quantifies the number of papers by FLOSS type. It points out that the most FLOSS type investigated in experiments for bug report severity prediction is Programming Tool (25 out of 27), closely followed by Application Software (21 out of 27). FLOSS classified as System Software are used in only 2 out of 27 papers. The chart in

Figure 8 quantifies the number of papers by BTS site. It shows up that all (27 out of 27) experiments published in the selected papers have extracted data from at least one FLOSS hosted on a Bugzilla site. In the same chart, it can be observed that only a few of them used FLOSS hosted on Google (3 out of 27) and Jira (2 out of 27) sites.



(a)



(b)

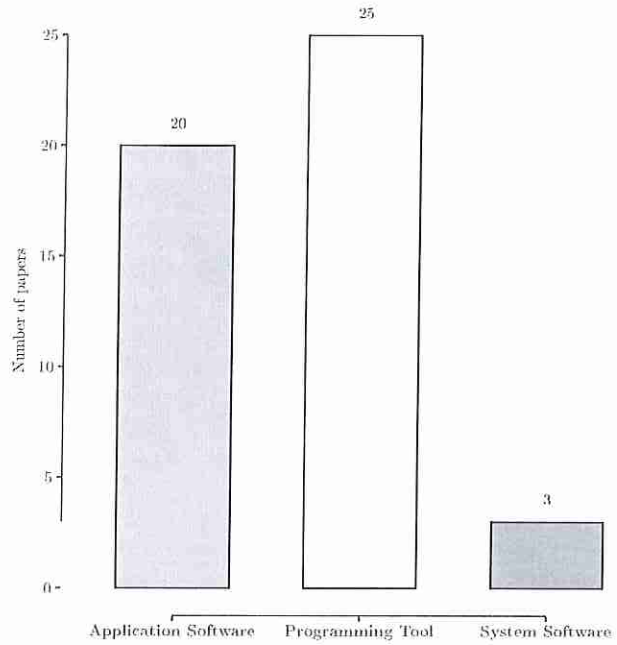
Fig. 6: (a) Paper distribution by year. (b) % of papers by source.

TABLE VI: FLOSS investigated in papers.

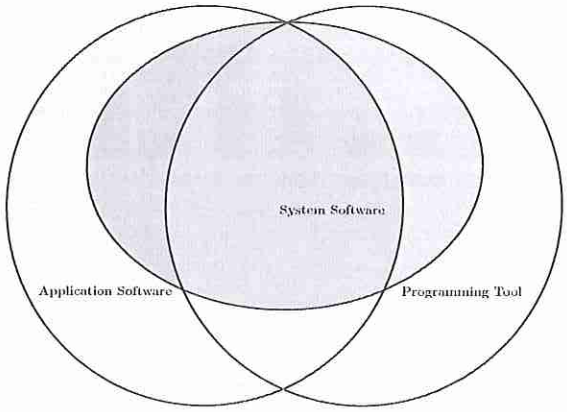
Project	Description	Category	BTS	URL (As of June 12, 2018)
Android	Mobile operating system	System software	Google	https://issuetracker.google.com
Chrominum	Web browser	Application software	Google	https://www.chromium.org/issue-tracking
Eclipse	Integrated Development Environment(IDE)	Programming tool	Bugzilla	https://bugs.eclipse.org/bugs/
FreeDesktop	Base platform for desktop 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/

TABLE VII: Paper distribution by FLOSS.

Project	2010	2011	2012	2014	2015	2016	2017	Total (%)
Eclipse	#1	#2	#3, #4, #5	#6, #7, #8, #10	#11, #12, #13, #14, #15	#17, #18, #19, #20, #21, #22, #23, #24	#25, #26, #27	92.6%
Mozilla	#1		#3, #4, #5	#6, #8, #9, #10	#12, #14	#16, #17, #19, #20, #21, #23, #24	#26, #27	70.4%
OpenOffice			#3	#8	#14	#19, #20		18.5%
Netbeans				#6, #8	#14	#20		14.8%
Gnome	#1	#2	#5					11.1%
Chrominum				#8	#14			7.4%
FreeDesktop				#8	#14			7.4%
Android							#25	3.7%
GCC						#20		3.7%
Hibernate							#27	3.7%
JBoss							#25	3.7%
Mongo-db							#27	3.7%
Spring							#27	3.7%
WineHQ						#16		3.7%

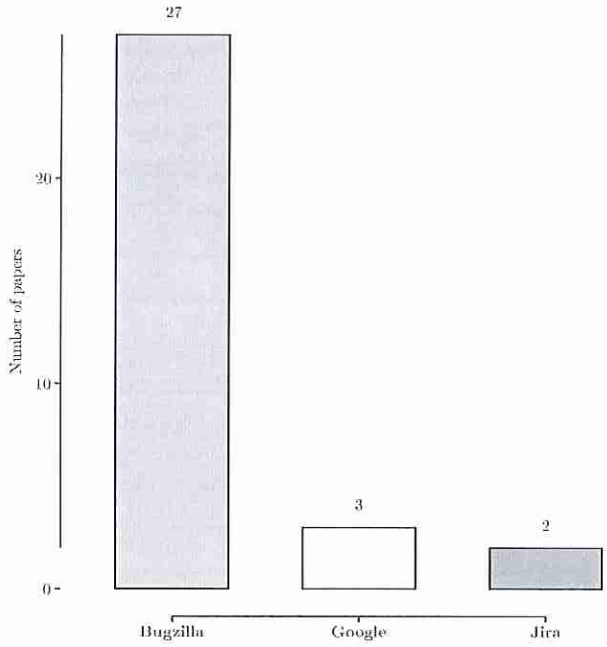


(a)

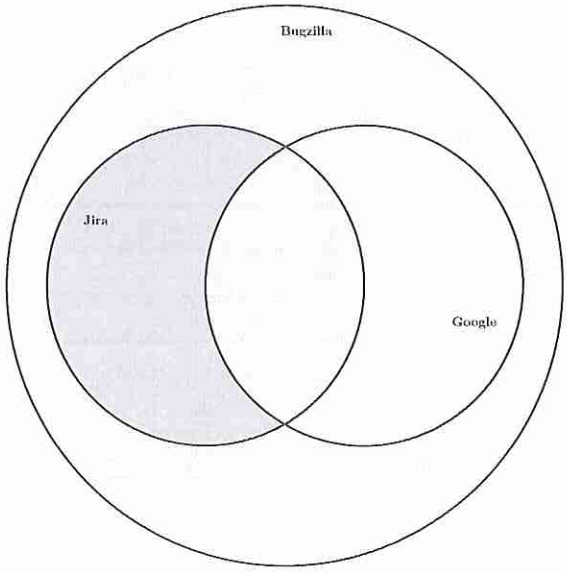


(b)

Fig. 7: (a) Paper distribution by FLOSS. (b) Paper distribution by FLOSS.



(a)



(b)

Fig. 8: (a) Paper distribution by BTS. (b) Paper distribution by BTS.

C. Has the prediction of severity of bug reporting been more commonly treated as a two-label, tree-label or multi-label prediction problem (RQ3)?

Table VIII shows prediction problem types faced by authors in selected papers for bug report severity prediction. It indicates that the most faced prediction problem type was multi-label (13 out of 27 - around 48%), and a significant number of papers (12 out 27 - around 44%) addressed two-label prediction problem as well. Only a few of them handled the tree-label problem.

TABLE VIII: Paper distribution by prediction problem type.

Problem Type	2010	2011	2012	2014	2015	2016	2017	Total(%)
Multi-label			#3, #5	#6, #9	#12	#16, #18, #19, #20, #22	#25, #26, #27	48.15%
Two-label	#1	#2	#4	#7, #8, #10	#13, #14, #15	#17, #21, #23		44.44%
Tree-label					#11	#24		7.41%

The chart in Figure 9 drills down the paper distribution by prediction problem type. It indicates that most of the papers (10 out of 27 - around 37%) address predicting bug report severity as an SNS problem. Moreover, the chart shows that 8 out of 27 (around 37%) addressed this problem as an MC and 5 out of 27 (around 18.5%) as an MCWD problem. A few papers regarded this problem as BNB (2 out of 27 - around 7%) or as SNSWD (2 out of 27 - around 7%).

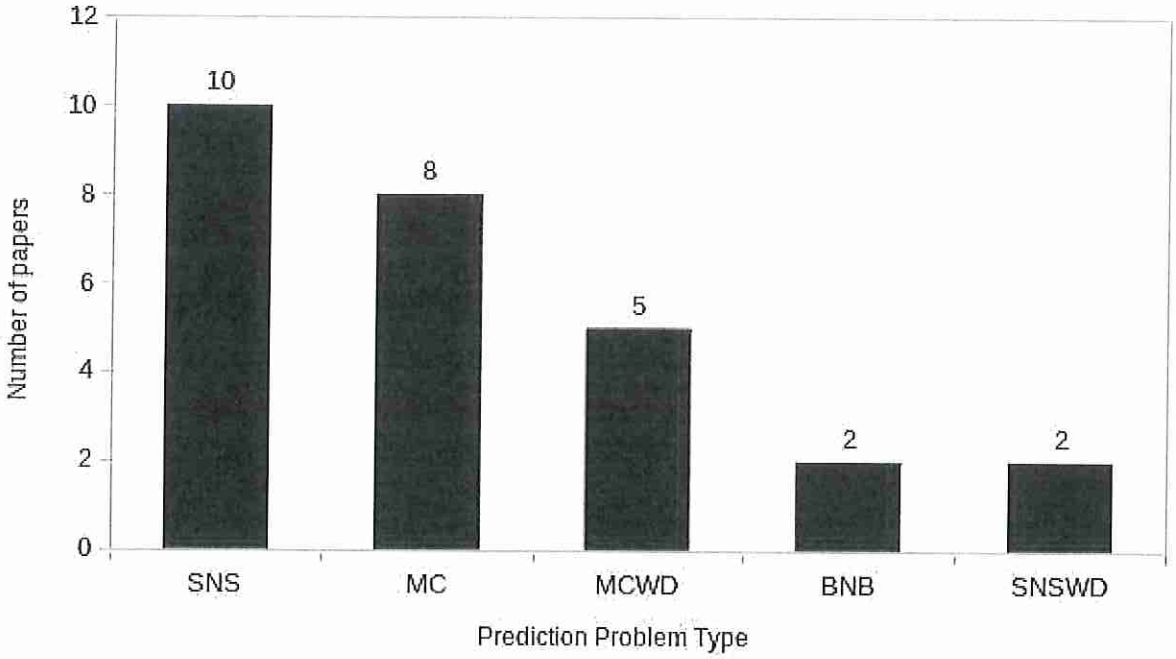


Fig. 9: Paper distribution by prediction problem type.

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

Table IX details the features ~~investigated~~ used in select ~~papers~~ ed studies. The details for each feature include a textual description; a category, according to Section IV-D; whether features are computed from others or not; and whether a feature is provided by Bugzilla, Jira and Google. Table X presents the feature distribution by papers. It can be observed that summary (19 out of 27 - 81.5%) and description (17 out of 27 - 63.0%) were the most common features used for bug report severity prediction.

The chart in Figure 10 groups features by data type in three categories: qualitative, quantitative discrete and quantitative continuous. It can be observed that most feature data type used for bug report severity prediction was qualitative.

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

Table XI shows the methods used in selected papers for bug report severity prediction. It can be observed that: (i) few papers utilized a feature selection method and; (ii) only filter method type was used by them.

Onde estão os footnotes

TABLE IX: Features descriptions.

Feature	Description	Category	Is Calculated?	Bugzilla[1]	Jira[2]	Google[3]
Attachments	Files (e.g., test cases or patches) attached to bugs.	Qualitative	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	No	Yes	No	Yes
Comment	Textual content appearing in the comments of bug report.	Qualitative	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	Yes	Yes	Yes	Yes
Component	Each product is divided into different components (e.g. Core, Editor, UI, etc).	Qualitative	No	Yes	Yes	Yes
Description	Textual content appearing in the description field of the bug report.	Qualitative	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	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	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	No	Yes	Yes	Yes[4]
Product	What general "area" the bug belongs to (e.g. Firefox, Thunderbird, Mailer, etc).	Qualitative	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	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	No	Yes	Yes	No
Severity	This indicates how severe the problem is – from blocker ("application unusable") to trivial ("minor cosmetic issue").	Qualitative	No	Yes	Yes	No
Summary	A one-sentence summary of the problem.	Qualitative	No	Yes	Yes	Yes
Summary weight	Calculated using information gain criteria	Quantitative Continuous	Yes	Yes	Yes	Yes

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

Table XII shows feature weighting models used for bug report severity prediction. It reveals that the most used model in papers was TF-IDF (9 out of 27 - around 33%).

Table XIII shows feature vector extraction models used for bug report severity prediction. It reveals that the most used model in papers was Bigrams (3 out of 27 - around 11%).

Table XIV Table shows feature vector similarity/dissimilarity functions used for bug report severity prediction. It reveals that the most used function in papers was BM25 and BM25ext with 3 out of 27 papers (around 3%) used them.

The chart in Figure 11 shows papers by text mining feature model type distribution. It points out that most used text mining method for bug report severity prediction was term model type (12 out of 27 - around 44%).

Isso não foi definido

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

Table XV shows ML algorithms used in selected papers for bug report severity prediction. It can be observed that the two most used ML algorithms used for bug report severity were k-NN and NB (12 out of 27 - around 44%).

The chart in Figure 12 shows ML algorithm types by paper distribution. It points out that the most used ML algorithm type for bug report severity prediction was probabilistic-based (21 out of 27 - around 77%).

TABLE X: Features used in selected papers.

Feature	2010	2011	2012	2014	2015	2016	2017	Total (%)
Summary	#1	#2	#3, #4, #5	#6, #7, #9	#11, #13	#16, #17, #20, #21, #23, #24	#25, #26, #27	81.5%
Description	#1		#3	#6, #7, #8, #10	#14, #15	#16, #17, #18, #20, #21, #23, #24	#25, #27	63.0%
Component			#3	#6, #8	#14	#16, #20, #21, #23, #24		33.3%
Product			#3	#6, #8	#14	#16, #20, #21, #23, #24		33.3%
Severity				#8	#14	#21, #23, #24		18.5%
Priority				#6, #8, #9	#14			14.8 %
Reporter						#21, #23, #24		11.1%
Description size				#8	#14			7.4%
Number in CC list				#8	#14			7.4%
Reporter blocking experience				#8	#14			7.4%
Reporter experience				#8	#14			7.4%
Reporter name				#8	#14			7.4%
Attachments				#7				3.7%
Bug fix time				#9				3.7%
Bug Id						#22		3.7%
CC list				#9				3.7%
Comment				#8				3.7%
Comment size					#14			3.7%
Complexity				#9				3.7%
Importance						#16		3.7%
Number of comments				#9				3.7%
Number of dependents				#9				3.7%
Number of duplicates				#9				3.7%
Platform					#14			3.7%
Report length				#7				3.7%
Summary weight				#9				3.7%

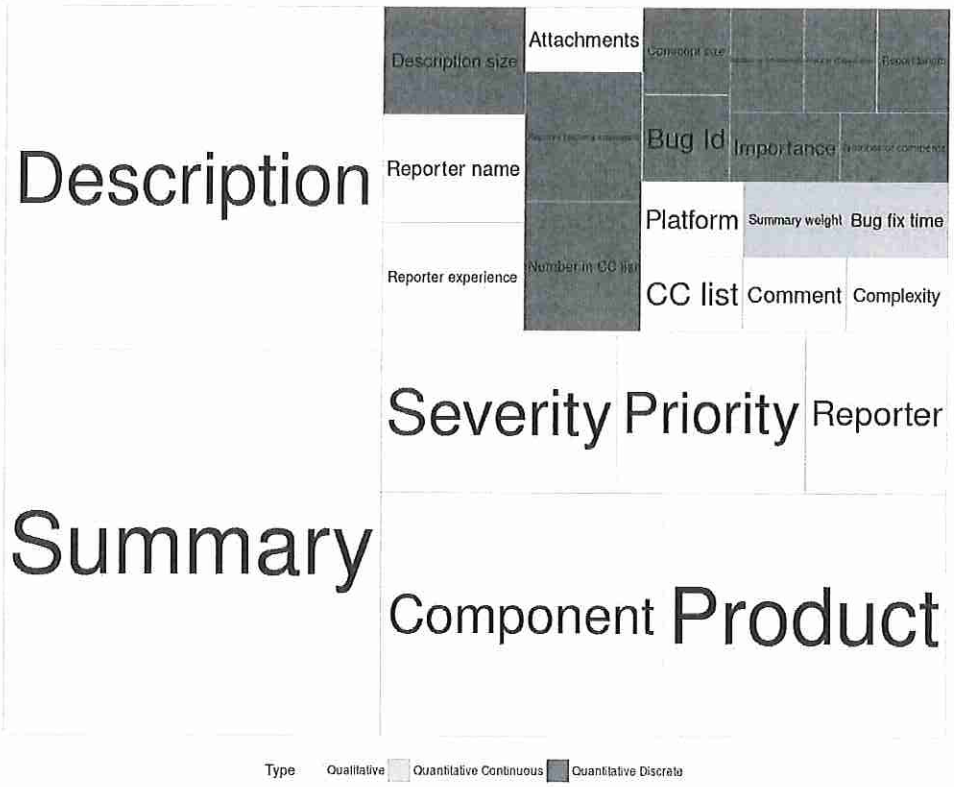


Fig. 10: Papers by feature data type.

TABLE XI: Paper distribution by feature selection methods.

Method	Category	2010	2011	2012	2014	2015	2016	2017	Total(%)
Information Gain	Filter			#4, #5		#13	#19	#27	18.52%
Chi-Square	Filter			#4	#10	#13			11.11%
Correlation Coefficient	Filter			#4					3.70%

TABLE XII: Paper distribution by feature vector weighting model.

Model	Category	2010	2011	2012	2014	2015	2016	2017	Total(%)
TF-IDF	Term		#2	#5	#10	#12, #13, #15	#17	#26, #27	33.33%
TF	Term		#2	#4					7.41%
Binary	Term		#2						3.70%

TABLE XIII: Paper distribution by feature vector extraction model.

Model	Category	2010	2011	2012	2014	2015	2016	2017	Total(%)
Bigrams	Character			#3	#10		#20		11.11%
Unigrams	Character			#3			#20		7.41%
Topic model	Concept						#20		3.70%

TABLE XIV: Paper distribution by feature vector similarity/dissimilarity function

Function	Category	2010	2011	2012	2014	2015	2016	2017	Total(%)
BM25ext	Term						#20		3.70%
BM25	Term			#3					3.70%

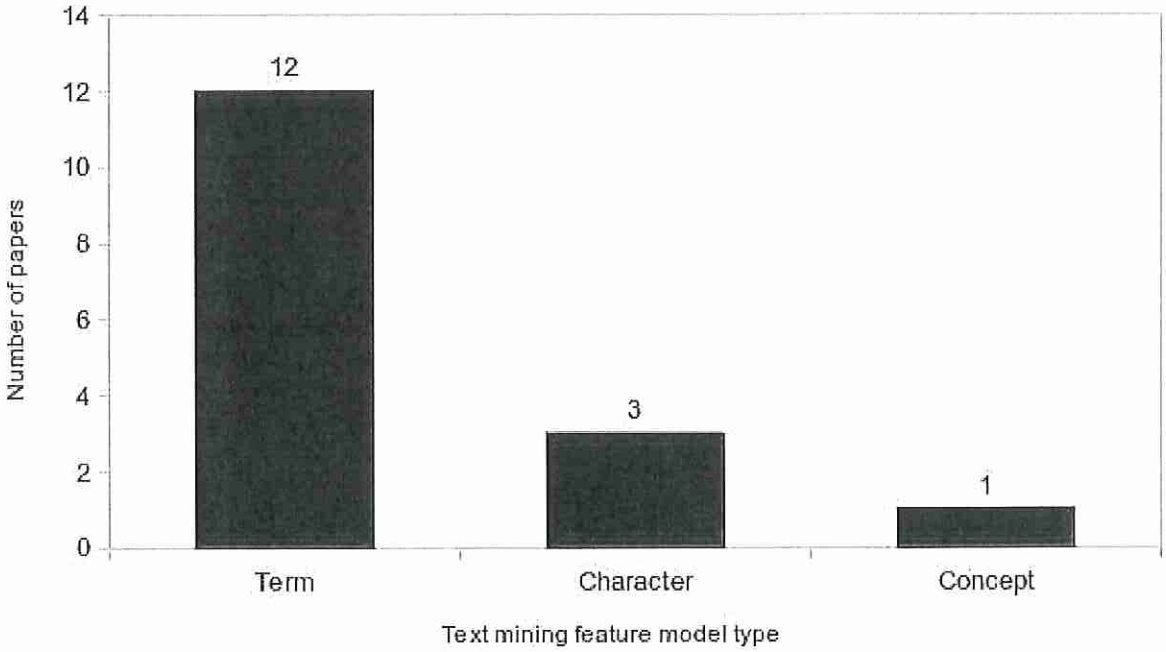


Fig. 11: Papers by text mining feature type

Discutir

TABLE XV: Papers distribution by ML algorithms

Technique	Category	2010	2011	2012	2014	2015	2016	2017	Total(%)
KNN	Distance		#2	#3	#6, #8, #9	#12, #15	#18, #19, #20, #22	#26	44.44%
NB	Probabilistic	#1	#2	#5	#6, #8, #9, #10	#11, #12	#17, #22	#26	44.44%
NBM	Probabilistic		#2	#4	#7	#12, #13, #15	#19, #21, #23, #24	#25	40.74%
SVM	Optimization		#2	#5	#9		#19, #22	#26, #27	25.93%
Random Forest	Searching			#5	#8	#14	#17, #22		18.52%
C4.5	Searching			#5	#8, #16				11.11%
Bagging Ensemble	Ensemble					#14, #16			7.41%
Decision Tree	Searching						#19, #22		7.41%
AdaBoost	Ensemble						#17		3.70%
RBF Networks	Optimization						#17		3.70%
RIPPER	Other			#5					3.70%
Zero-R	Other				#8				3.70%

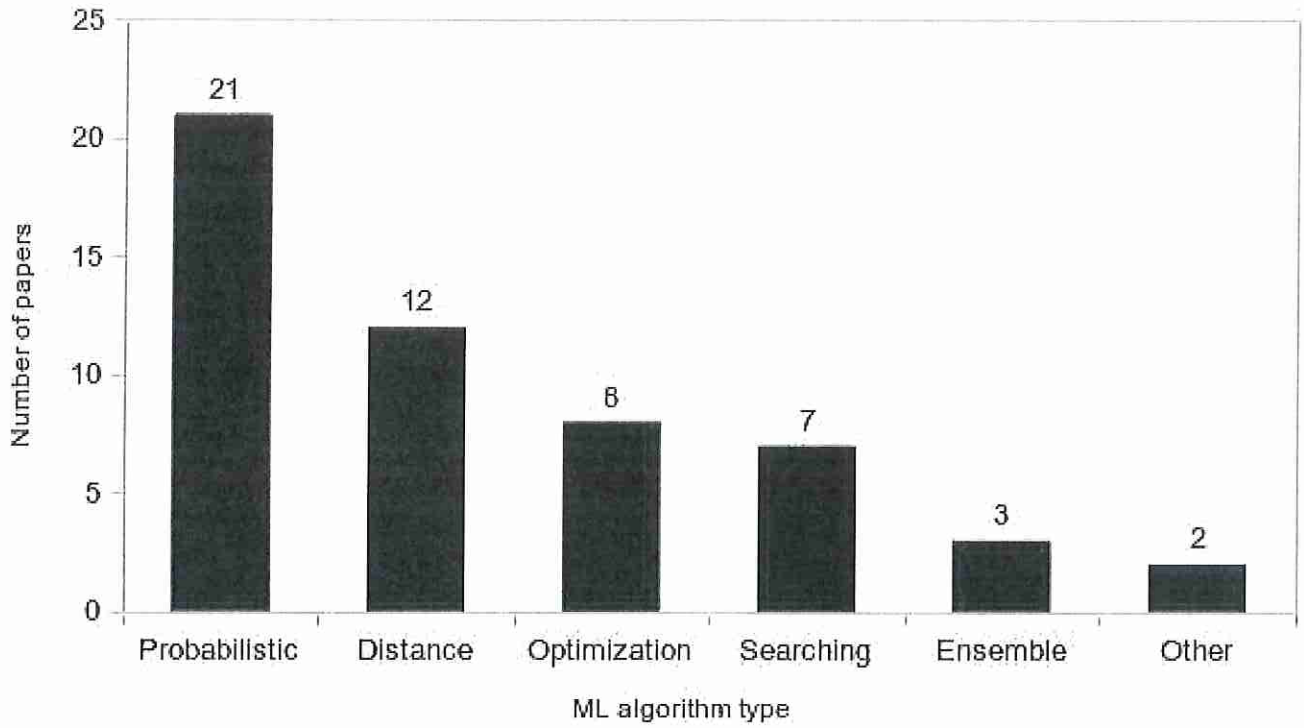


Fig. 12: Papers by ML algorithm categories

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

Table XV shows evaluation measures used in selected papers for bug report severity prediction. It can be observed that the most used evaluation measure for bug report severity prediction was precision (21 out of 27 - around 77%).

The chart in Figure 13 shows paper distribution by ML performance evaluation measure types. It can be observed that the most used evaluation measure type in bug report severity prediction was a single class (21 out of 27 - around 70%).

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

Table XVII shows sampling methods used in selected papers for bug report severity prediction. It can be observed that sampling method preferred by authors of these papers was 10-Fold CV (10 out of 27 - around 37%).

The chart in Figure 14 shows papers distribution by sampling method type. It can be observed that the most sampling method used in bug report severity prediction was simple re-sampling (15 out of 27 - around 55%).

Varias não explicadas

TABLE XVI: Papers distribution by evaluation measures.

Measure	Type	2010	2011	2012	2014	2015	2016	2017	Total(%)
Precision	Single class	#1		#3, #5	#6, #8, #9, #10	#12, #13, #14, #15	#16, #18, #20, #21, #22, #23, #24	#25, #26, #27	77.78%
Recall	Single class	#1		#3, #5	#6, #8, #9, #10	#12, #14	#16, #18, #20, #21, #22, #23, #24	#25, #26, #27	70.37%
F-measure	Single class	#1		#3, #5	#6, #8, #9, #10	#12, #14	#18, #20, #21, #22, #23, #24	#25, #26, #27	66.67%
Accuracy	Multi-class			#5	#6, #8, #9, #10	#11, #13, #15	#16, #17	#26	40.74%
AUC	Summary	#1	#2	#4	#7, #10				18.52%
ROC	Graphical	#1			#10				9.52%
MRR	Other						#6, #20		7.41%
Effectiveness Ratio@20%	Other					#14			3.70%
Krippendorff's Alpha Reliability	Other						#19		3.70%

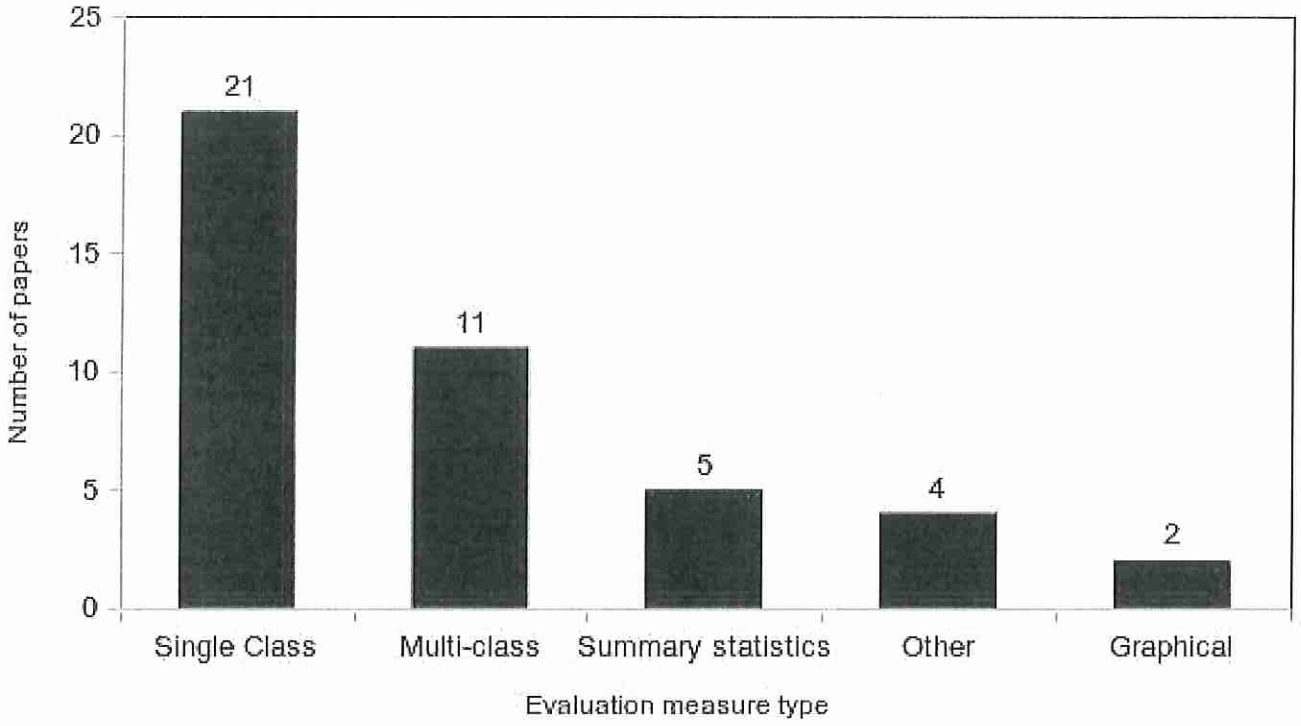


Fig. 13: Papers by evaluate measure types.

TABLE XVII: Paper distribution by sampling methods.

Sampling Method	Type	2010	2011	2012	2014	2015	2016	2017	Total(%)
10-Fold CV	Simple re-sampling		#2	#4, #5	#7	#14	#16, #17, #23	#25, #26, #27	37.04%
Stratified 10-Fold CV	Simple re-sampling				#8, #10				11.11%
Hold-out	No re-sampling						#17		3.70%
SMOTE	Simple re-sampling						#22		3.70%
03-Fold CV	Simple re-sampling						#22		3.70%
05-Fold CV	Simple re-sampling					#13			3.70%

não explicado

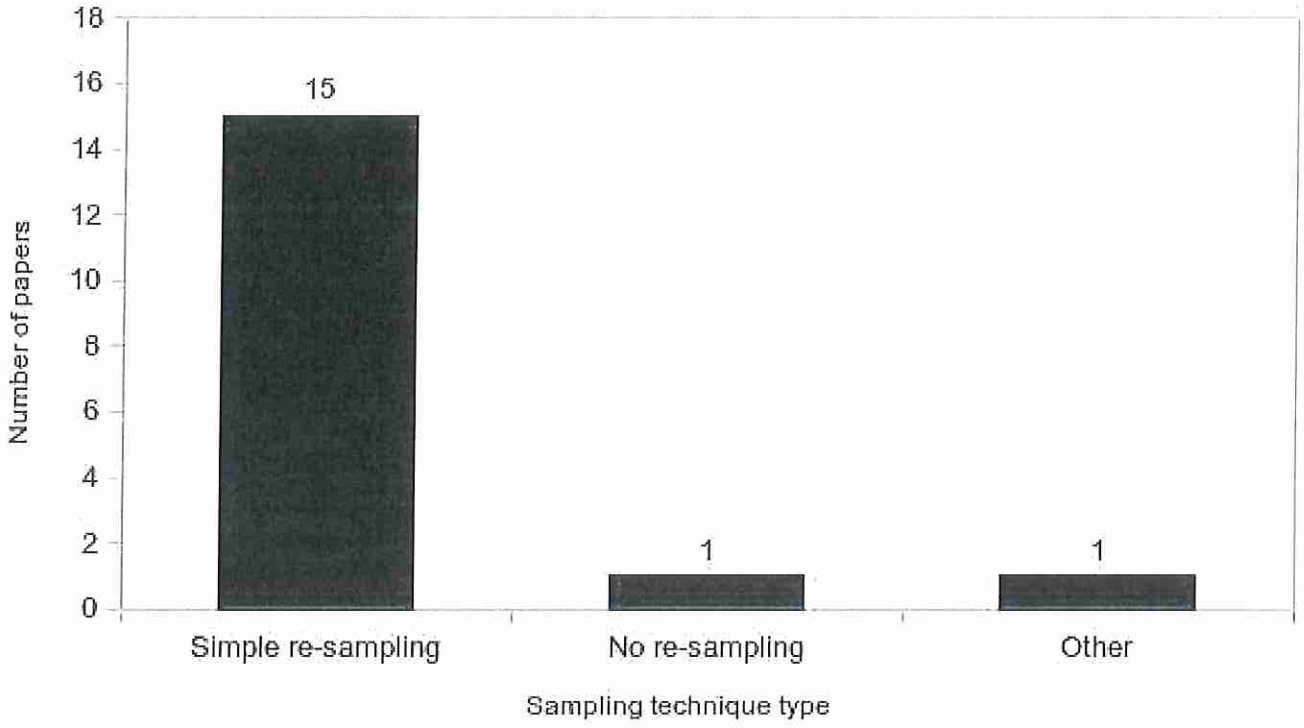


Fig. 14: Papers distribution by sampling method types.

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

Table XVIII shows statistical tests used in selected papers for bug report severity prediction. It can be observed that the statistical tests most used by authors of these papers were T-test and Wilcoxon signed rank test (7 out of 27 - around 25%).

TABLE XVIII: Paper distribution by statistical tests.

Statistical Test	Type	2010	2011	2012	2014	2015	2016	2017	Total(%)
T-test	Parametric				#6	#12	#20, #22, #23	#25, #27	25.93%
Wilcoxon signed rank test	Parametric and non-parametric				#6, #8	#12, #14	#20, #23	#25	25.93%
Mann-Whitney U test	Non-parametric							#26	3.70%
Proportion test	Parametric							#27	3.70%
Shapiro-Wilk test	Parametric							#25	3.70%

The chart in Figure 15 shows papers distribution by statistical test type. It can be observed that the most statistical test type used in bug report severity prediction was parametric (9 out of 27 - around 33%).

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

Table XIX shows software tools used in selected papers for bug report severity prediction. It can be observed that the software tool most used by authors of these papers was RapidMiner (5 out of 18 - around 18%).

The chart in Figure 16 shows software tools distribution by select papers. It can be observed that the most used type of software tool for bug report severity prediction was FLOSS (11 out of 27 - around 40%).

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

Table XX shows the solution types proposed in selected papers for bug report severity prediction. It can be observed that the most solution type was off-line (26 out of 27 - around 93%).

- baselines
- effectiveness

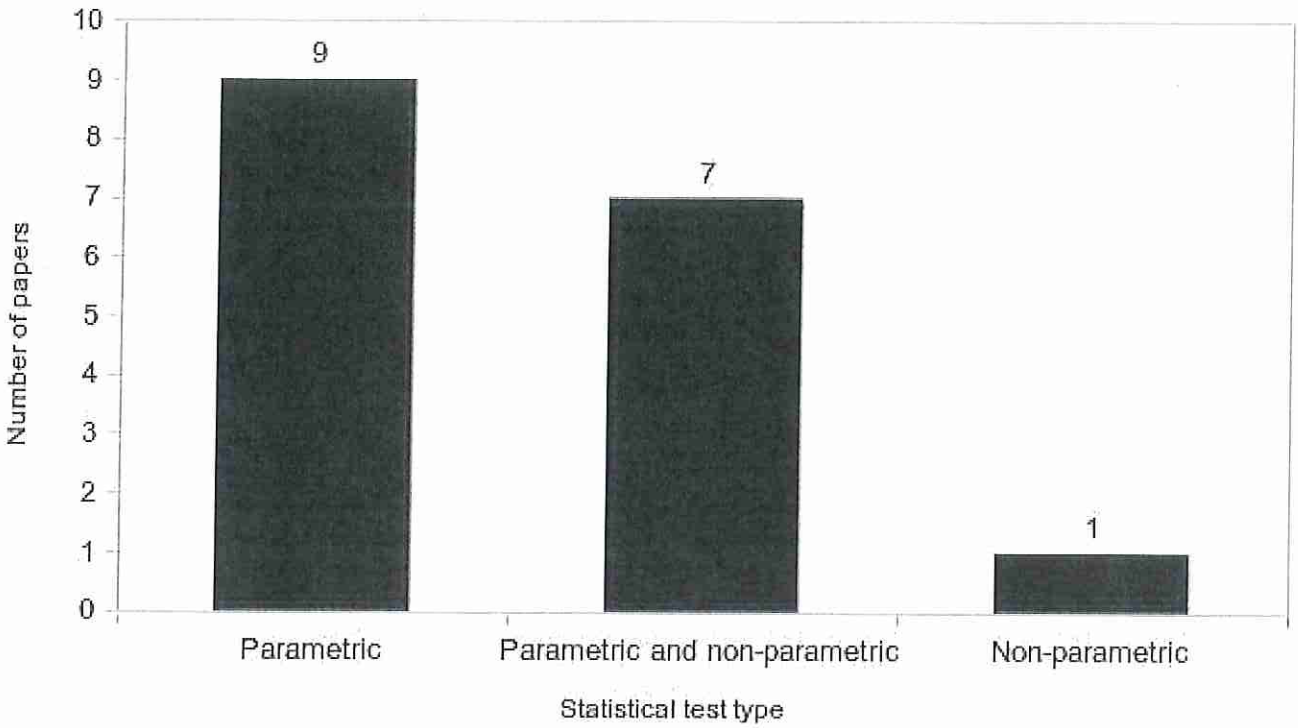


Fig. 15: Papers distribution by statistical test types.

TABLE XIX: Software tools used in selected papers.

Software	Type	2010	2011	2012	2014	2015	2016	2017	Total(%)
RapidMiner	CSS			#5	#9	#13, #15		#26	18.52%
WEKA	FLOSS		#2		#14, #16		#19		14.81%
R	FLOSS				#6	#12		#25	11.11%
NLTK	FLOSS				#10		#20, #23		11.11%
TMT	FLOSS				#6		#20		7.41%
Statisca	CSS			#5	#9				7.41%
Ruby	FLOSS	#1							3.70%
OpenNLP	CSS			#3					3.70%
WVTool	FLOSS			#4					3.70%
WordNet	CSS				#10				3.70%
CoreNLP	FLOSS						#25		3.70%

TABLE XX: Solution types proposed in selected papers.

Solution	2010	2011	2012	2014	2015	2016	2017	Total(%)
off-line	#1	#2	#4, #5	#6, #7, #8, #9, #10	#11, #12, #13, #14, #15	#16, #17, #18, #19, #20, #21, #22, #23, #24	#25, #26, #27	96.30
on-line			#3					3.70

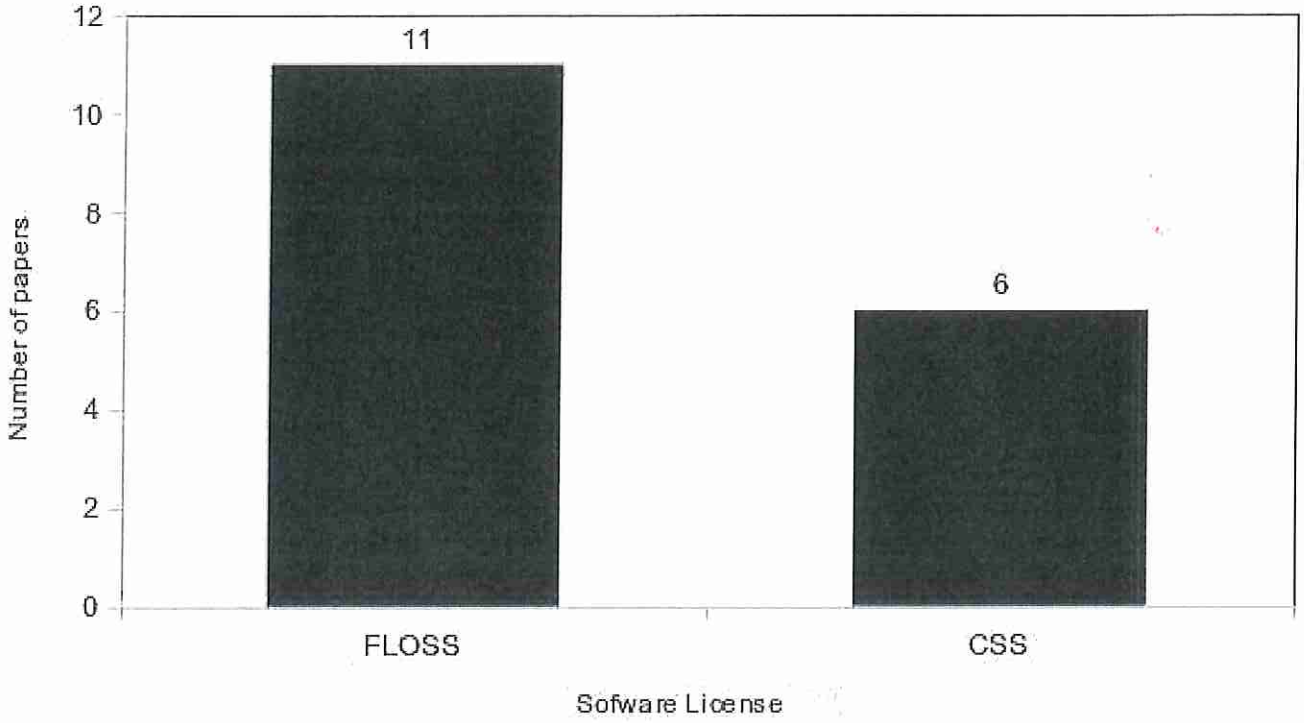


Fig. 16: Paper distribution by software tool types.

VI. Discussion

This section discusses the lessons learned from conducted study and recommends possible research directions for the bug severity prediction problem.

The results indicate that research focusing on bug report severity prediction is still recent with the vast majority of the studies developed in the last four years. As well as, there is no particular place to publish, or there is no well-established forum for debating this topic. Information Technology, Mining Software Repositories, and Software Engineering publication vehicles seem to be more receptive to papers of bug report severity prediction.

Despite the number of projects (14 in the total) used as bug reports source, the results show a lack of others prominent FLOSS, such as Linux Kernel, Ubuntu Linux, and MySQL. Furthermore, there is a concentration of research efforts over on two projects: Eclipse (approximately 88% of papers) and Mozilla (approximately 70% of papers). Many authors reason that such concentration is needed to benchmark their results with Lamkanfi et al. [4], [5], which are often considered the state of the art in this researching area.

It can be observed that there is a strong focus on bug reports from Bugzilla because it hosts both projects mentioned before. Therefore, papers slightly investigated two others bug trackers cited in studies: Jira, which tracks bugs for more than 80% Apache Foundation's projects[5]; and Google Issue Tracker, which tracks bugs for many internal and external Google Inc. projects[6]. No paper investigated Github, an outstanding collaborative website which hosted more than 67 million of projects (including many FLOSS)[7]. Hence, it may remain a green field for researching in bug report severity prediction.

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 depends on who writes it [4] [5]. So, the probability of writing bug reports more accurate is greater for technical users than end-users. It may infer that the most of efforts to severity prediction run over "good" bug reports.

The researchers worked on bug report severity prediction as separate classification problem categories. The most faced by them was SNS which does not consider the default severity level (frequently named "normal") as a valid class. Doing in this way, they have argued that they would avoid noisy and unreliable data. However, just ignoring "normal" severity level, experiments can throw away a large percentage of the reports. Saha et al. [30] proposed a partial solution to this problem by

[5]http://www.apache.org/index.html#projects-list
[6]https://developers.google.com/issue-tracker/
[7]https://octoverse.github.com/

creating a clean dataset of two manners: (i) inspecting manually the dataset or (ii) selecting only reports where severity has been changed after the initial submission.

Summary and Description were the most used features for bug report severity prediction. They were utilized respectively in 81.5% and 63.0% of selected papers. These results suggest that proposed solutions are heavily based on unstructured textual information, which leverages advanced text mining and natural language processing methods to extract useful data. Product and Component were other two features reasonably used in papers (33% of them). It may suppose that bug reports collected from separate parts of the same FLOSS affect ML algorithms outcomes differently.

Regarding ~~of~~ available feature data types in bug reports, more than 50% of them are qualitative. SVM and Neural Networks, two popular and essential ML algorithms in modern machine learning [16], do not work with qualitative data [14]. These combined facts bring an extra challenging to use them for bug reports severity prediction. When an input data set has qualitative data, the categorical values must be converted into numeric values before ~~to~~ applying ML algorithms.

The only three used feature selection approaches for bug severity prediction were Information Gain, Chi-Square, and Correlation Coefficient. Yang et al. [24] demonstrated their effectiveness to extract potential severe and no-severe indicators and improve the prediction performance in over half the studied cases. Therefore, it can realize that very few papers (5 out of 27) used at least one method to select features. Furthermore, these papers notably only investigated filter feature selection methods. This approach has two drawbacks [14]: (i) it does not take into account redundancy between features, and (ii) it does not detect dependencies between them.

Results show that vast proposed solutions based on unstructured text information (summary and description). Despite this fact, rare papers reported utilizing some text mining method. The text mining methods most used by these few was bigrams, TF-IDF, and BM25.

Traditional and well-known ML algorithms run in most of experiments: k-NN [47] or NB [48](around 44% of them), NBM [49] (around 40% of them), and SVM [50](around 25% of them). However, they perform well, and most of the times a paper outperformed the previous one; the state-of-~~art~~ the ML algorithms may improve these outcomes in bug report severity prediction. Deep learning, for instance, performs very well when classifying ~~for~~ audio, text, and image data [51].

To evaluate the performance of ML algorithms, most of papers worked with precision and recall, both single class measure. Many of these papers employed F-measure, a harmonic mean of precision an recall, as well as. In a real scenario, a typical BTS can store many more bug reports with a normal, major, or minor than bug reports with a blocker, critical or trivial severity level [44]. Imbalanced data scenario, as the previous one, strongly skews precision and recall measures [52]. To minimize this distortion, Tian et al. [6], instead, recommend measuring performance using inter-rater agreement based metrics, such as Cohen's Kappa [53] or Krippendorff's Alpha [54]. Despite its public issues in an imbalanced dataset [20], around 40% of papers still used accuracy to measure ML performance. AUC, an alternative used by about 18% of papers, seems more robust than accuracy in imbalanced data conditions. Like Kappa and Krippendorff's Alpha, it considers the class distribution on performance evaluation of ML algorithms for bug report severity prediction.

Most of papers used the common 10-fold Cross-Validation technique. About 11% employed the Stratified 10-Fold Cross-Validation, a simple variation of standard 10-fold Cross-Validation which is useful when the class distribution of data is skewed [20]. It is worth to observe that only one paper used SMOTE, considered "de facto" resampling method in imbalanced data scenario [55]. Furthermore, most of papers, which used at least one sampling technique, utilized a simple re-sampling one. No paper investigated a method of multiple resampling. Repeated k-fold Cross Validation, a multiple resampling technique, may obtain more stable estimates of algorithm's performance and enhance replicability of results [20].

About 25% of papers employed at least one statistical test: T-test or Wilcoxon Signed Rank test [56]. No paper investigated Analysis of Variance (ANOVA), a robust ranked-based test recommended by Kitchenham et al. [57] for Empirical Sofware Engineering. Most used statistical test was parametric (around 33% of them), only one paper use a non-parametric test. Parametric tests require that samples follow a statistical distribution (e.g., normal), which is not guaranteed to occur in practice when comparing ML models [19].

It is worth to note that, despite their importance, few papers regarded statistical tests to compare ML algorithms performance using statistical significance. In this sense, Japkowicz et al. [20] call attention that to determine whether one algorithm is better than the other by simple exam of superiority or inferiority of means is not advisable.

RapidMiner was the preferred software tool to run experiments for bug severity prediction. However, the owner software company provides a limited free version (w/ 30 days trial), the basic license of this tool costs $2,500 (user/year)[8], which may be prohibitive for many research groups. Preventing them from reproducing some experiments reported in papers. On the other hand, a significant portion of experiments run over FLOSS tools (40% of papers). It worth to observe that only one paper used R and no paper used Python, considered two favorite free programming language to implement machine learning solutions[9].

The most proposed solutions for bug report severity prediction were off-line. Only the work of Tian et al. [3] claimed that the solution presented by them would be fast enough to be embedded, for example, in a web browser. The vast of papers provided off-line machine learning solutions. Huge number of bug reports, high rate of bug reports per day and poor learning

[8]https://rapidminer.com/pricing/

[9]https://www.infoworld.com/article/3186599/artificial-intelligence/the-5-best-programming-languages-for-ai-development.html

reeserever/ frase soft

training performance seems to be the main challenges to deliver online solutions for bug report severity prediction. As opposed to ML algorithms that are trained offline, Continuous Learning technology [58] may provide the necessary tools to overcome these limitations.

VII. Conclusions and Future Works

A systematic mapping study suggests a structure of the type of research reports and results that have been published by categorizing them and often gives a visual summary of its results [9]. These results aid to identify gaps in a research area, becoming a basis for future research, as well as, to provide a guide to position new research activities adequately [8].

The systematic mapping presented in this paper captured the current state of research on bug report severity prediction, characterized related problems and identified the main approaches employed to resolve them. These objectives were reached by conducting mapping of existing literature. In total, the study identified 27 relevant publications and analyzed them along 12 dimensions. The previous analysis resulted in following general considerations:

- Research on bug report severity prediction in FLOSS seems to be recent and active with a vast number of papers published from 2014.
- Researchers concentrated their focus on few FLOSS, namely Eclipse and Mozilla, both hosted by a Bugzilla site. Notably, they did not consider relevant FLOSS projects like as Linux Kernel, Spark, MySQL, among others. Moreover, they briefly investigated Jira and did not investigate in any way GitHub, two others important BTS players.
- The bug report prediction is a single problem, but researchers faced it considering five kinds of the classification problem. Two of five kinds not considered default severity level (often "normal") as a valid class. 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. Even so, the same authors do not recommend just discarding these bug reports. ?
- Most of the solutions 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 important role to deliver these solutions. Suggesting that modern text mining methods may advance the knowledge or improve results yielded so far.
- Despite a large number of features raised by the application of text mining activities (tokenizing, stop word removal and stemming) on summary and description. Feature selection methods called little attention of researchers. Few papers employed such techniques in ~~its~~ experiments for bug severity prediction.
- Most papers did not explicitly relate using of text ming methods. Few of them reported ~~utilizing~~ to use mainly TF-IDF (for feature vector weighting), bigram (for feature extraction) and BM25 (for verifying similarity).
- It seems that researchers have adopted a conservative posture regarding the use of ML algorithms. Most papers worked with at least one of the following well-known and traditional supervised algorithms: k-NN, Näive Bayes, and a variation of latter, the Näive Bayes Multinomial. Thus, experiments did not address more modern approaches such as deep and continuous learning.
- Researchers evaluated the performance of ML algorithms in their proposed solutions using, in most of the situations, precision, recall, and f-Measure. They are three common measures 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.
- Few papers reported using a resampling approach to improving the accuracy of ML algorithms. All methods related belongs to simple resampling category, including, the most used, k-fold cross-validation. Japkowicz et el. [20] suggest~~s~~ 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 and also with others observing only the means of measures adopted. Few of them reported utilizing statistical tests to do that, which is a recommended practice in empirical software engineering [57].
- Researchers preferred to use FLOSS tools to perform their analyses. Although, a reasonable number of papers, which described their computation environment for experiments, 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 [59] is a problem intrinsic to supervised ML algorithms may make it difficult to turn these solutions to online.

Although the select~~ed~~ papers have done valuable work to advance knowledge in bug report severity prediction, the outcomes presented by this systematic mapping suggest that there are excellent research opportunities for further advancement in this topic and to produce better results.

Such as ?

(falta uma discussão científica p/ fechar

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] 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.
[7] N. V. Chawla, "Data Mining for Imbalanced Datasets: An Overview," in *Data Mining and Knowledge Discovery Handbook*. Boston, MA: Springer US, 2009, pp. 875–886.
[8] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," 2007.
[9] 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.
[10] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2010.
[11] R. Feldman and J. Sanger, *Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. New York, NY, USA: Cambridge University Press, 2006.
[12] 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.
[13] 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.
[14] P. Flach, *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. New York, NY, USA: Cambridge University Press, 2012.
[15] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.
[16] S. Marsland, *Machine Learning: An Algorithmic Perspective, Second Edition*, 2nd ed. Chapman & Hall/CRC, 2014.
[17] G. Williams, *Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery*, 2011.
[18] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
[19] 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.
[20] N. Japkowicz and M. Shah, *Evaluating Learning Algorithms: A Classification Perspective*. New York, NY, USA: Cambridge University Press, 2011.
[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] 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.
[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

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

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

[47] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?" in *Proceedings of the 7th International Conference on Database Theory*, ser. ICDT '99. London, UK, UK: Springer-Verlag, 1999, pp. 217–235.

[48] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[49] A. M. Kibriya, E. Frank, B. Pfahringer, and G. Holmes, "Multinomial naive bayes for text categorization revisited," in *AI 2004: Advances in Artificial Intelligence*, X. Webb, Geoffrey I.and Yu, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 488–499.

[50] N. Christianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge: Cambridge University Press, 2000, vol. 18, no. 6.

[51] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[52] L. A. Jeni, J. F. Cohn, and F. D. L. Torre, "Facing imbalanced data–recommendations for the use of performance metrics," in *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*, Sept 2013, pp. 245–251.

[53] A. Ben-David, "Comparison of classification accuracy using cohen's weighted kappa," *Expert Systems with Applications*, vol. 34, no. 2, pp. 825 – 832, 2008.

[54] K. Krippendorff, "Computing krippendorff's alpha-reliability," *Departamental Papers (ASC)*.

[55] A. Fernandez, S. Garcia, F. Herrera, and N. V. Chawla, "On the unreliability of bug severity data," *Journal of Artificial Intelligence Research.*, vol. 61, pp. 863–905, Apr. 2018.

[56] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Dec. 2006.

[57] 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.

[58] Z. Chen and B. Liu, "Lifelong machine learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 10, no. 3, pp. 1–145, 2016.

[59] 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.