Learning to Predict Long-Lived Bug Report Severity Prediction on Free/Libre Open Source Software

Qualification Exam

Ph.D. Student: Luiz Albero Ferreira Gomes
Principal Advisor: Prof. Mario Lúcio Côrtes
Co-advisor: Prof. Ricardo da Silva Torres
Institute of Computing
UNICAMP
August 29, 2018

Abstract

Software evolution and maintenance activities in today Free/Libre Open Source Software rely primarily on information extracted from reports registered in bug tracking systems. The severity level attribute of a bug report is considered one of the most critical variables for planning these activities. For example, this attribute measures the impact the bug has on the successful execution of the software system and how soon a bug needs to be addressed by the development team. An incorrect assignment of severity level may generate a significant impact on the maintenance process. Nevertheless, methods to assign severity level remains essentially manual with a high degree of subjectivity. Because it depends on the experience and expertise of whom have reported or reviewed the bug report, it may be considered a quite error-prone process. Due to its evident importance, both business and academic community have demonstrated a great interest in this topic, which has been associated with an extensive investigation towards the proposal of methods to predict the bug report severity automatically. We show in a systematic mapping review annexed to this qualification exam document that traditional machine learning methods and text mining techniques have been playing a successful role in these efforts. This review also reveals that two important challenges in this area are imbalanced data and high dimensionality in datasets used for prediction. Besides, the proposed approaches have not been considered the lifecycle duration and the temporal context of bug reports. Studies show that short-lived bug reports represent around 50% of bug reports in some relevant FLOSS projects, and they are closed within seven days for many reasons (e.g., fixed, duplicated, or invalid). Thus, efforts to predict severity level of short-lived bug reports seems meaningless and useless. Conversely, the number of long-lived bugs is high and over 90% of them adversely affect the user's experience. The same studies suggest that many long-lived bugs could be fixed quickly through careful triaging and prioritization, if developers could, for example, predict their severity, in advance. This research work aims to bridge this gap by proposing a learning model for long-lived bug report severity that addresses the temporal context for them which for severity prediction for this group of bug reports seems paramount. We propose two approaches to build this learning model: (i) develop new feature selection methods or extend existing feature selection methods which address the temporal context issues of a long-lived bug report, imbalanced data, and high dimensionality in bug reports datasets used to severity prediction; and (ii) develop new data-driven approaches for long-live bug severity prediction which take into account the temporal context of a long-lived bug report, imbalanced data and high dimensionality in bug reports datasets.

I. Introduction

Bug Report Tracking Systems (BTS) have a major role in the evolution and maintenance process in many software development settings, both in Closed Source Software (CSS) and Free/Libre Open Source Software (FLOSS) scenarios. This is especially true in FLOSS, which is 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 several bug reports [1].

A user interacts with a BTS often through a simple mechanism called bug report form. This form enables him to request changes, to report bugs or to ask for support in software product [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 development team member will review this report and, if it is not refused for some reason (e.g., request duplication), he or she will provide additional information in the bug report form, indicating, for example, its priority and the person who will be responsible for the bug report.

Typically, the number of bug reports in large and medium software FLOSS projects is frequently very large [3]. Therefore, efforts have been made to implement intelligent software assistants to help developers and maintenance personnel in defining more accurately the attribute values in a bug report form [4].

Severity level shifts throughout bug report *lifecycle* may have an adverse effect on the planning of maintenance activities. For example, the maintenance team could be demanded to address less significant bug reports before most important ones. Thus, severity level information is recognized as a critical variable in the equation to estimate a prioritization of bug reports [2]. It defines how soon the bug reports need to be addressed [5]. 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], [2], [5]. Consequently, it is a time-consuming and error-prone process with a high degree of subjectivity.

In a comprehensive systematic mapping review of recent research efforts on automatically bug report severity prediction, we point out that these efforts have overlooked bug reports lifecycle duration (Annex I). Saha et al. [6] examined seven FLOSS projects and noted that around 50% of bug reports are closed within seven days for many reasons (e.g., fixed, duplicated or invalid). Thus, efforts to predict severity level of these short-lived bug reports seems to be useless. Also, the same study reveals that there are a considerable number of long-lived bugs in each system and over 90% of them adversely affect the user's experience. Besides, the study suggests which many long-lived bugs could be fixed quickly through careful triaging and prioritization, if developers could, for example, predict their severity, in advance.

The definition of long-lived is subjective since the time threshold for deciding whether a bug is long-lived or short-lived could vary across projects, persons, or studies. Saha et al. [6], for instance, consider that a long-lived bug should survive over at least two release cycles of a product which corresponds at a minimum one year in the FLOSS investigated in this paper. In this interval, it is very likely that the long-lived bug report information will evolve making them more realistic. Because of this, considering this temporal context is essential for severity predicting of a long-lived bug report. In Figure 1, for example, as a user includes (e.g., writes a new comments) or updates information (e.g., change the description) during the bug report A lifecycle (from t_0 to t_{n+1}), and as bug report repositories also evolve over time by the inclusion and update of bug reports, the severity of a bug report A could change from trivial to critical.

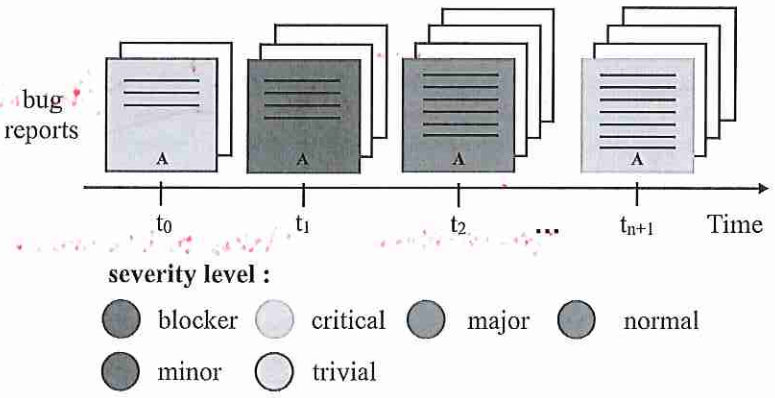

Fig. 1: Bug report temporal evolution.

Furthermore, our mapping review reveals the current research efforts have strongly relied on traditional techniques in Machine Learning (ML), Feature Selection (FS) and Text Mining (TM) whose yielded results can be improved, for example, by applying state-of-the-art of these techniques. Besides, this review also

reveals that two important challenges in this area are imbalanced data and high dimensionality in datasets used for prediction.

This research work aims to bridge this gap by proposing a learning models for long-lived bug report severity that addresses the temporal context for them which for severity prediction for this group of bug reports seems paramount. We propose two approaches to build this learning model: (i) develop new feature selection methods or extend existing feature selection methods which address the temporal context issues of a long-lived bug report, imbalanced data and high dimensionality in bug reports datasets used to severity prediction, and (ii) develop new data-driven approaches for long-live bug severity prediction which take into account the temporal context of a long-lived bug report, imbalanced data and high dimensionality in bug reports datasets.

The proposal is organized as follows. Section II describes the research problem and hypothesis. Section III defines the goals of our work. Section IV presents the background concepts. Section V discusses related works that are relevant to our research. Section VI details our thesis proposal. Section VII describes our research method. Section VIII presents the scheduled time-line. Finally, Annex I presents our systematic mapping review of bug report severity prediction in FLOSS projects.

II. Problem and Hypothesis

Bug report severity prediction is typically modeled as a classification problem. After the preprocessing step, bug reports attributes will be converted into a feature vector. Formally, in a training dataset with a set of N bug reports or documents, each document is represented as an m-dimensional vector $x = (\{x_1, x_2, ..., x_m\})$, where $x_j \in X_j$ corresponds to the value of random variable $X_j (j = 1, 2, \ldots, m)$. The set $X = X_1 \times X_2 \times \ldots \times Xm$ denotes the *input space*. This set contains all possible inputs to the learning algorithm. Classification tasks require that the machine learning produces an output. The *output space* is defined in a similar way, where $y \in Y$ denotes the value of the output random variable Y. The output space can also have more than one dimension and express this is analogous to the input space. Variables X_j are often referred to as *features* or *dimensions*, **x** as *sample* or *instance*, and Y as *label* or *class*.

Considering temporal contexts of long-lived bug reports, both the m-dimensional vector x and the output random variable y may evolve in time. To represent correctly such temporal context, x will be defined as $x_t = \{\{x_1^{t_0}, x_2^{t_0}, ...x_m^{t_0}\}, \{x_1^{t_1}, x_2^{t_1}, ...x_m^{t_1}\}, \ldots, \{x_1^{t_n}, x_2^{t_n}, ...x_m^{t_n}\}\}$ and y will be defined as $y_t = \{y^{t_0}, y^{t_1}, ..., y^{t_n}\}$ where $n = (0, 1, 2, 3, ...)$ and $T = \{t_0, t_1, t_2, ..., t_n\}$ be set of moments on temporal space when both test and training documents were create or updated. Besides, the set of classes $y^t \in (c_1, c_2, ..., c_i)$, where $(i = 2, 3, ..., k)$ depends on the BTS in which FLOSS is hosted. Typically k ranges between five and six classes (e.g., blocker, critical, major, normal, minor, trivial).

In the real world, bug reports training and test datasets are imbalaced [7] and high-dimensional [8]. The input space (x_t) in the temporal context of long-lived bug report makes the high-dimensional problem more difficult since now we could have to consider n sets of x.

Therefore, we can state our problem formally as:

> *the long-lived bug report severity prediction problem could be formally dea fined as prediction the severity level of long-lived bug reports considering its temporal context and simultaneously minimizing the measurable effects related to imbalanced and high-dimensional data.*

This problem will be investigated considering the following research hypotheses:

H1. The temporal context of long-lived bug report leads to the performance improvement of classification systems used to bug report severity prediction.

H1. The feature selection considering the temporal context of long-lived bug reports mitigate the effects of imbalanced data and high-dimensional in classification systems used to severity prediction.

H3. The temporal context modeling of long-lived bug reports using data-driven methods leads to the performance improvement of classification systems used to bug report severity prediction.

III. Goals and Contributions

The general goal of this research project is to build a learning model that allows predicting the severity of long-lived bug reports, which considers the temporal context of the bug report and which address two common problems in bug reports dataset: imbalanced data and high-dimensional. As a result of the research carried out, the following objectives are also envisioned:

G_1 Study of the state of the art in bug report severity prediction in FLOSS projects, problems, and opportunities (completed).

$G_{1.1}$ Write a systematic mapping review on bug report severity prediction in FLOSS projects (Annex I).

$G_{1.2}$ Identify the main problems on bug report severity prediction (Annex I, pages 22 and 23).

G_2 Select one or more problems and advance the state of the art in this area (partially completed).

$G_{2.1}$ The following problems were selected:

a. Prediction severity considering the temporal context of long-lived bug reports.
b. Imbalanced temporal context datasets in long-lived bug report severity prediction.
c. High dimensionality in temporal context bug reports datasets in long-lived bug report severity prediction.

$G_{2.2}$ Develop new feature selection methods or extend existing feature selection methods which address the temporal context isssues of a long-lived bug report, imbalanced data and high dimensionality in bug reports datasets used to severity prediction.

$G_{2.3}$ Develop new data-driven approaches for long-live bug severity prediction which take into account the temporal context of a long-lived bug report, imbalanced data and high dimensionality in bug reports datasets.

The contributions of this research project for computer science are:

1) New learning models that will use the temporal evolution information of bug reports for long-lived bug report severity prediction which so far has not been considered in any proposed solution for this problem.
2) New learning models based on novels feature selection methods which address the temporal context of a long-lived bug report, imbalanced data, and high-dimensionality in bug reports datasets used for bug report prediction.
3) New learning models based on novels data-driven approaches to long-lived bug report prediction which address the temporal context of a bug report, imbalanced data, and high-dimensionality in bug reports datasets.

The contributions of this research project for FLOSS maintenance area are:

1) New learning models which effectively address long-lived bug report and improve the maintenance process of FLOSS projects. This important because long-lived bugs may propagate through more than one major releases of a product, which may affect the user experience negatively for many major versions [6].

IV. Background

Our systematic review (Anenx I, section II) provide a explanation of a rich set of concepts and terminology related to bug report severity prediction area. This section briefly introduce some specific concepts, which were not addressed in this review.

A. Deep Learning

The proposed approaches in the literature have been strongly adopted traditional ML algorithms for bug report severity prediction (Annex I, Table XIII, page 18). However, these algorithms need careful feature engineering and substantial domain expertise to design a feature extractor that transformed the raw data

into a feature vector from which a classifier could predict a class to an input instance. To automatically classification, deep learning enables a machine to read raw data and to discover its representations. It has been proved to be very good at discovering complex structures in high-dimensional [9] data.

Deng and Yu [10] define deep learning as "a sub-field within machine learning that is based on algorithms for learning multiple levels of representation to model complex relationships among data".

The Convolutional Neural Network (CNN) and the Recurrent Neural Network (RNN) are two mainstream architecture for sentence and document modelling. A CNN can be defined as a feedforward neural network, which consists of an input and an output layer, as well as multiple hidden layers. It is capable of capturing the global and the local features and significantly enhancing the efficiency and accuracy [11]. On the other hand, a RNN is suitable for modeling sequential data because connections between nodes form a directed graph along a sequence. There are loops and memories in RNN to remember former computations. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are two examples of RNN [11].

B. Word Embedding

Bag-of-Words was the vector extraction model most used for bug severity prediction in the literature (Annex I, Table XII, page 18). This model was based on counting frequencies of occurrences of shot symbols sequences of length up to N (N-grams). The number of possible N-grams is on the order of V^N, where V is the vocabulary size, so taking into account a context of more than a handful of words would require very large training corpora [9].

A principal component of the neural-network approach is the use of embeddings - representing each feature as a vector in a low dimensional space. Word embeddings are a class of techniques which represents individual words as real-valued vectors in a predefined vector space. It learns a vector representation of words by mapping semantic information into a geometric word embedding space. Such models, the vector representation w of a given the word is usually learned through a fixed context window W. Each word is mapped to one vector, and the vector values are learned in a way that resembles a neural network. There is two-word embeddings representation:

- **Word2Vec [12]** is a statistical method for learning a standalone word embedding from text corpus which involves analysis of learned vectors and the exploration of vector math on the representation of words.
- **GloVe [13]** is a statistical method which captures semantic information from the fixed context window and global corpus statistics through the word co-occurrence probabilities.

C. Genetic programming algorithm

Few approaches for bug severity prediction used some method for feature selection, all of them traditional (Annex I, page 99). Recent and promising methods for feature selection based on Genetic Programming (GP) have been successfully used in Automatic Document Classifying (ADC) [14]–[16]. This technique provides a framework for automatically creating a working computer program from a high-level problem statement of the problem. GP achieves this goal of automatic programming by genetically breeding a population of computer programs using the principles of Darwinian natural selection and biologically inspired operations [17]. As a population-based evolutionary computation technique, GP typically follows these steps [14]:

1) Generate an initial population of individuals composing of primitive functions and terminals of the problem.
2) Iteratively performs the following sub-steps until a stopping criterion is met:
   a) Evaluation: each individual is executed, and its fitness is calculated based on a predefined fitness function.
   b) Selection: select one or two individuals from the population with a probability based on fitness to participate in the evolution step.

c) Evolution: create new individuals for the new population by applying the following genetic operators with specific probabilities: (i) copy the selected individuals to the new population (reproduction); (ii) create new offsprings by recombining randomly chosen parts from two selected programs (crossover); and (iii) create a new offspring program by randomly mutating a randomly chosen part of one selected program (mutation).

3) Return the program with the highest fitness as the best solution.

Applying GP to solving the problem has to specify a terminal set, a function set, a fitness function, a stopping criterion and control parameters such as population size, crossover and mutation probabilities.

V. Related Work

This section presents includes more related works to those presented in our mapping review. Works presented here has large potential to be adapted to our proposed learning model.

A. Graph-based text representation

The approaches presented in this section typically model a text document as a graph, in which a vertex presents a term from the text and an edge represents a relationship between terms.

Dourado [18] proposed a Bag Textual Graph (BoTG), an accurate graph-based text representation model that combines term counting with also locality principles such as proximity and the order terms within a text, in an efficient general-purpose vector space representation. The objective is to model textual documents as graphs and to project them in a vector space that has lower dimensionality and higher efficiency than the Bag-of-Words model, with comparable accuracy to the graph models but with more general applicability. BoTG is targeted to be used in a wide range of applications.

Defferrard et al. [19] present a formulation of CNN in the context of spectral graph theory, which provides the necessary mathematical background and efficient numerical schemes to design fast localized convolutional filters on graphs. Importantly, the proposed technique offers the same linear computational complexity and constant learning complexity as classical CNNs, while being universal to any graph structure. Experiments on MNIST and 20NEWS demonstrate the ability of this deep learning system to learn local, stationary, and compositional features on graphs.

Kipf et al. [20] present a scalable approach for semi-supervised learning on graph-structured data that is based on an efficient variant of convolutional neural networks, which operate directly on graphs. According to authors, their model scales linearly in number of graph edges and learns hidden layer representations that encode both local graph structure and feature nodes.

B. Temporal context aware methods

Salles et al. [21] provided evidence of the existence of temporal effects in three textual datasets, reflected by variations observed over time in class distribution, in pairwise class similarities, and in the relationships between terms and classes. They also that these temporal effects affect each analyzed dataset differently and that they restrict the performance of each considered text classification algorithm.

Salles et al. [22] proposed a machine learning methodology to learn the *temporal weighting function* (TWF) automatically and they also propose new strategies to incorporate the TWF into ADC algorithms, which we call *temporally-aware* classifiers. Experiments showed that fully-automated temporaly-aware classifiers achieved significant gains (up to 17%) when compared to their non-temporal counterparts, even outperforming some state-of-the-art algorithms (e.g., SVM) in mosts cases, with large reductions in execution time.

C. Feature Selection Methods

Tran et al. [14] investigated the use of GP for feature construction and selection on high-dimensional classification problems. Different combinations of the constructed and/or selected features were tested and compared on seven high-dimensional gene expression problems, and different classification algorithms are were to evaluate their performance. The results showed that the constructed and/or selected feature sets can significantly reduce the dimensionality and maintain or even increase the classification accuracy in most cases.

Viegas et al. [16] propose a novel feature selection strategy based on Genetic Programming, which is resilient to data skewness issues, in other words, it works well with both, balanced and unbalanced data. The proposed strategy aims at combining the most discriminative features sets selected by distinct feature selection metrics in order to obtain a more effective and impartial set of the most discriminative features, departing from the hypothesis that distinct feature selection metrics produce different (and potentially complementary) feature space projections. This new strategy resulted in a massive reduction in the number of features (up to 65% in a textual dataset) without the loss of effectiveness.

D. Data Driven Methods

Zhang et al. [23] offer an empirical exploration of the use of character-level convolutional networks (ConvNets) for text classification. They constructed several large-scale datasets to show that character-level convolutional networks could achieve state-of-the-art or competitive results.

Lai et al. [24] applied RNN to capture contextual information and max-pooling layer that judges which words play critical roles in text classification to capture the key components in texts. The experiments results show that the proposed method outperforms the state-of-the-art on several datasets, particularly on document-level datasets.

Yang et al. [25] used word-embeddings for feature extraction and CNN for classifying, and achieved significantly better performance compared to baselines such as SVM and TF-IDF in Twitter election classification.

Jiang et al. [26] outperformed traditional ML algorithms using Deep Believe Network (DBN) for feature extraction and softmax regression for classifying texts from Reuters-21,578 and 20-Newsgroup. Also, approaches based on deep learning have been used with relative success in the text classification and feature selection in textual bases. For example, LSTM [27], ConvNet [23], and RNN [24]. In addition, our proposal will consider temporal evolution information issues on deep learning modelling and evaluation.

VI. Proposal

As previously state, we aim to conceive learning models for long-lived bug severity prediction with two main requirements to attain our objectives: (i) they must be able to handle data that changes over time; (ii) they must be resilient to high-dimensionality in datasets used for prediction; (iii) they must be resilient do imbalanced data in datasets used for prediction.

Figure 2 presents the main modules of our proposed learning model to meet de requirements outlines above. The module 1 will be in charge of downloading bug reports from FLOSS repositories (e.g., based on Bugzilla, Github, and Jira) and store them in a raw data format into a disk. After that, the module 2 will preprocess these bug reports, which involves type checking, normalizing, fix and imputation activities [28], and transform them in a consistent data format suitable for machine learning classifiers.

In order to organize bug reports data, as well as, its relations, we will investigate methods that can learn from dynamic graphs to our learning model. As a starting point, we propose to extend the BoTG [18] to represent the temporal context of long-lived bug reports. An advantage of this model for our problem is to organize data in low-dimensionality.

After organizing data of our learning model, we investigate features selection methods which are imbalance data and high dimensionality resilient. Initially, we will extend features selection methods based

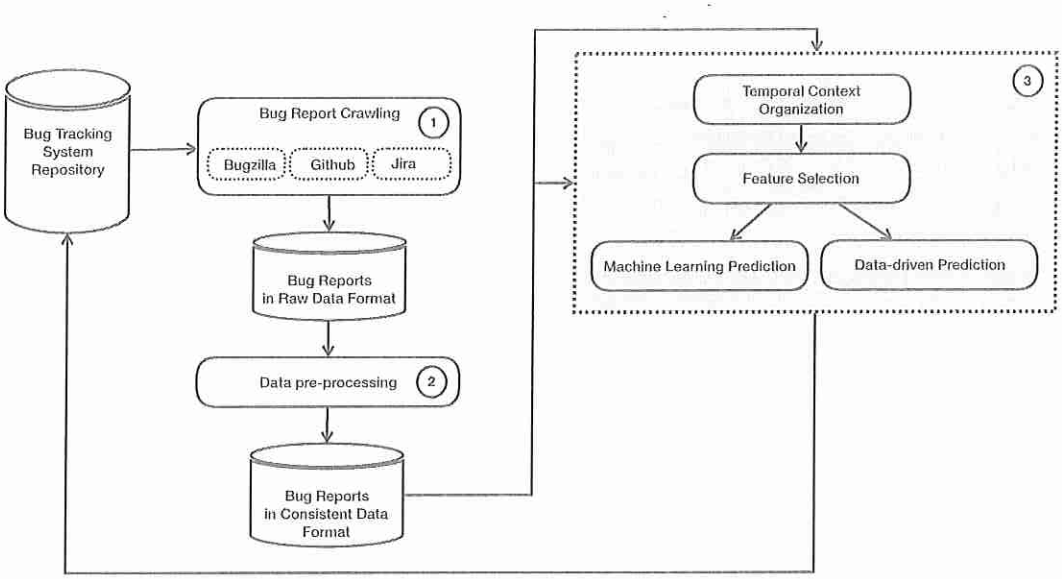

Fig. 2: Proposed learning model.

on GP [14]–[16] to consider the temporal context of long-lived bug reports. Such methods have been well-succeed to address imbalanced and high-dimensionality in textual datasets for document classification.

To accomplish the prediction task of our learning model (module 3), we will investigate two approaches (i) machine learning methods; and (ii) data-driven methods. Proposed approaches reviewed in our systematic mapping have ~~been~~ commonly used traditional machine learning methods for bug report severity prediction. Initially, we employ these traditional methods in our learning model. ~~Doing this~~, we ~~would~~ ~~enable~~ to compare ~~more clearly~~ our yielded results with results published in the literature. Also, we investigate and used data-driven methods in prediction task. Well-succeed approaches for text classifying using data-driven methods, such deep learning, motivate us to ~~investigate~~ this research avenue [25], [26], [27], [23], and [24].

VII. Research Method

The primary research method of this research project is based on empirical experimentation. As in typical methodologies used in ML experiments, we follow the next steps:

- **Data Extraction**: this step in the experimental research encompasses selecting FLOSS projects to serve as the data source, studying and interpreting its data structure, and finally extracting relevant data from its repository (feature extraction).
- **Data preprocessing**: Raw data previously collected from selected FLOSS repositories is not correctly structured to serve as input to ML algorithms, they were not in a consistent data format. The classical way to address this problem is to run preprocessing procedures to extract, organize, and structure relevant features out of the raw data.
- **Training and testing**: Training and testing steps start with partitioning the already preprocessed dataset in two disjoint subsets: a subset from training, typically with 80% of the bug reports, and a subset for testing, with the remaining 20% of the bug reports.
- **Evaluation**: this step in the experimental research encompasses the evaluation of experiments results and validation utilising statistical tests (Annex I, page 99).

A. Experimental Setup

Datasets: We will train and test our proposed model on datasets from FLOSS projects outlined in Table VII in our systematic mapping review (Annexed I). These relevant FLOSS projects are hosted in

distinct BTS (Bugzilla, Google, and Buzilla) and their bug reports are free available providing datasets with different properties to validate our learning model.

ML algorithms: We will build our proposed model using traditional ML algorithms outlined in Table XIII in our systematic mapping review (Annexed I). As starting point, we will investigate the four most used algorithms for bug report severity prediction: KNN, Näive Bayes, Näive Bayes Multinomial e SVM.

Evaluation metrics: We will evaluate our proposed model using metrics outlined in Table XIV in our systematic mapping review (Annexed I). As starting point, we will evaluate our learning model performance using the three most used metrics: precision, recall e f-measure.

Data driven algorithms: As starting point, will use two data-driven algorithms based on deep learning family considered mainstream in document modeling and well-succeed applied in text classification: CNN and RNN.

Baselines: We compare our learning model to the approaches proposed in papers outlined in Table IV in our systematic mapping review (Annexed I).

VIII. Project Timeline

Table I describes the project timeline that contemplates the doctoral degree requirements and research activities.

TABLE I: Project time line

Activity	2016	2017		2018		2019		2020
	2	1	2	1	2	1	2	1
(a) Literature review								
(b) Study of ML and TM Methods								
(c) Implementation of crawlers of bug reports								
(d) Development of scripts to run and analyze ML and TM experiments								
(e) Write a Systematic Mapping Review								
(f) Qualification Examination								
(g) Study of Genetic Programming and Deep Learning								
(h) Design, implement execute and evaluate experiments								
(i) Publish papers								
(j) Defense of Thesis								

The research steps are commented next:

b. Study of main ML and TM methods, as well as methodologies to effective using these methods (completed).

c. Implementation of Java Program to bug reports extracting from Bugzilla and Jira repositories, and converting them to a consistent format (completed).

d. Designing and implementation of experiments to learn and to test traditional ML algorithms to predict bug severity from bug reports collected from Bugzilla and Jira repositories. These experiments evolved the complete machine learning pipeline: data collecting, data pre-processing, training, and testing (completed).

e. Write a systematic mapping review of bug report severity prediction (Annex I) which yielded a paper that will be submitted to a journal (completed).

g. Study of main GP and Deep Learning methods which can be applied to our problem.

h. Design, implement, execute and evaluate experiments to validate our proposed our learning model.

i. Publish the results of our research in journals and conferences.

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] 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.

[3] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonckz, "Comparing mining algorithms for predicting the severity of a reported bug," *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*, pp. 249–258, 2011.

[4] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated prediction of bug report priority using multi-factor analysis," *Empirical Software Engineering*, vol. 20, no. 5, pp. 1354–1383, oct 2015.

[5] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," *Proceedings - International Conference on Software Engineering*, pp. 1–10, 2010.

[6] R. K. Saha, S. Khurshid, and D. E. Perry, "Understanding the triaging and fixing processes of long lived bugs," *Information and Software Technology*, vol. 65, pp. 114 – 128, 2015.

[7] 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.

[8] 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.

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436 EP –, May 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14539

[10] L. Deng and D. Yu, *Deep Learning: Methods and Applications*. Hanover, MA, USA: Now Publishers Inc., 2014.

[11] A. Karatzoglou and B. Hidasi, "Deep learning for recommender systems," in *Proceedings of the Eleventh ACM Conference on Recommender Systems*, ser. RecSys '17. New York, NY, USA: ACM, 2017, pp. 396–397.

[12] Y. Goldberg and G. Hirst, *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers, 2017.

[13] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *In EMNLP*, 2014.

[14] B. Tran, B. Xue, and M. Zhang, "Genetic programming for feature construction and selection in classification on high-dimensional data," *Memetic Computing*, vol. 8, no. 1, pp. 3–15, Mar 2016.

[15] B. Xue, M. Zhang, W. N. Browne, and X. Yao, "A survey on evolutionary computation approaches to feature selection," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 4, pp. 606–626, Aug 2016.

[16] F. Viegas, L. Rocha, M. Gonçalves, F. Mourão, G. Sá, T. Salles, G. Andrade, and I. Sandin, "A genetic programming approach for feature selection in highly dimensional skewed data," *Neurocomputing*, vol. 273, pp. 554 – 569, 2018.

[17] N. Nedjah, A. Abraham, and L. de Macedo Mourelle, *Genetic Systems Programming: Theory and Experiences*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[18] Ícaro Calvancate Dourado, "Bag of Textual Graphs," Master's thesis, University of Campinas, Campinas, Brazil, 2016.

[19] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. USA: Curran Associates Inc., 2016, pp. 3844–3852. [Online]. Available: http://dl.acm.org/citation.cfm?id=3157382.3157527

[20] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2016. [Online]. Available: http://arxiv.org/abs/1609.02907

[21] S. Thiago, R. Leonardo, G. M. André, A. J. M., M. Fernando, M. Wagner, and V. Felipe, "A quantitative analysis of the temporal effects on automatic text classification," *Journal of the Association for Information Science and Technology*, vol. 67, no. 7, pp. 1639–1667.

[22] T. Salles, L. Rocha, F. Mourão, M. Gonçalves, F. Viegas, and W. Meira, "A two-stage machine learning approach for temporally-robust text classification," *Information Systems*, vol. 69, pp. 40 – 58, 2017.

[23] X. Zhang, J. J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," *CoRR*, vol. abs/1509.01626, 2015.

[24] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15. AAAI Press, 2015, pp. 2267–2273.

[25] X. Yang, C. MacDonald, and I. Ounis, "Using word embeddings in twitter election classification," *CoRR*, vol. abs/1606.07006, 2016. [Online]. Available: http://arxiv.org/abs/1606.07006

[26] M. Jiang, Y. Liang, X. Feng, X. Fan, Z. Pei, Y. Xue, and R. Guan, "Text classification based on deep belief network and softmax regression," *Neural Computing and Applications*, vol. 29, no. 1, pp. 61–70, Jan 2018.

[27] D. S. Sachan, M. Zaheer, and R. Salakhutdinov, "Investigating the working of text classifiers," *CoRR*, vol. abs/1801.06261, 2018.

[28] E. de Jonge and M. van der Loo, "An introduction to data cleaning with R," *Statistics Netherlands*, p. 53, 2013. [Online]. Available: http://cran.r-project.org/doc/contrib/de_Jonge+van_der_Loo-Introduction_to_data_cleaning_with_R.pdf