# 1 Project Overview

Bug Report Tracking Systems(BTS), such as Bugzilla or Jira, have played a major role in the maintenance process in many software development settings, both in Closed Source Software (CSS) and in Open Source Software (OSS). This fact is especially true in OSS projects – characterized by many users and developers with different levels of expertise spread out the world – who might create or be responsible for dealing with several bug reports. A user interacts with a BTS often through a simple mechanism called bug report form[14]. This kind of form enables users to request changes, to report bugs, or to ask for support in a software product. Initially, he or she should inform a short description, a long description, a type (e.g., bug, new feature, improvement, and task), and an associated severity level (e.g., blocker, critical, major, minor, and trivial). Subsequently, a development team member will review this request and, in case it is not refused for some reason (e.g., bug report duplication), he or she will complete the information in bug report form, indicating, for example, its severity and assigning the person responsible for the bug report. [1]. In this context, my project proposes a machine learning web application for bug severity prediction based on the bug description filled by users when they open a bug in a BTS.

## 1.1 Problem Statement

The severity level information is recognized as a critical variable in the equation to estimate a prioritization of bugs. It defines how soon the bug report needs to be addressed. However, the severity level assignment remains mostly a manual process that relies only on the experience and expertise of the person who has opened the bug report [19]. As a consequence, 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 OSS projects is frequently very large. Severity level shifts throughout the bug report lifecycle may harm the planning of maintenance activities. For example, the maintenance team could be assigned to address less significant bug reports before the most important ones.

# 2 Solution Statement

The general purpose of my project is to develop an intelligent ML-based web assistant to help developers and maintenance team to predict the severity level of a bug (blocker, critical, major, minor, or trivial). To meet the goal, the tasks that should be executed are the following:

1. Collect bug reports data from a Bugzilla BTS;
2. Pre-process the description text using text mining techniques (e.g. tokenization).
3. Train and test a machine learning classifier to predict bug severity level based on bug report description;
4. Build and deploy a web application to extract a bug report description from a Bugzilla BTS and predict the bug severity level.

## 2.1 Datasets and Inputs

I'll use a dataset with 2395 bug reports, collected from Mozilla project and available in the project GitHub repository[1], to train and test the predicting model. The Mozilla project encompasses several open-source projects, such as Firefox, Thunderbird, and Bugzilla. Furthermore, it is well-established, has a considerable number of registered bug reports, uses standard repositories, and were being studied in academic research [7, 8, 14]. Table 1 describes the features of the dataset that will used for bug report severity prediction. Each feature in this table has a description, a category.

Figure 1 shows the label distribution in the Mozilla dataset, which will be used to train and test the prediction model. It will be split into a training set (75% out of 2395) and into a testing set (25% out of 2395). As one

[1] https://github.com/gomesluiz/bug-severity-predictor/blob/main/data/raw/mozilla_bug_report_data.csv

| Feature | Description | Category | Target Label |
|---|---|---|---|
| Bug id | Bug report identifier. | Quantitative Discrete | No |
| Description | Textual content appearing in the description field of the bug report. | Unstructured Text | No |
| Severity | Indicates how severe the problem is – from blocker (“application unusable”) to trivial (“minor cosmetic issue”). | Qualitative Ordinal | Yes |

Table 1: The features and its descriptions of dataset used for bug severity prediction. The target label of prediction model is the **Severity** feature

can observe, the label distribution in the dataset is unbalanced. Therefore, to maintain the proportion of labels in training and testing sets, I will use the stratify splitting and class weighting[10] to mitigate the unbalancing impact.

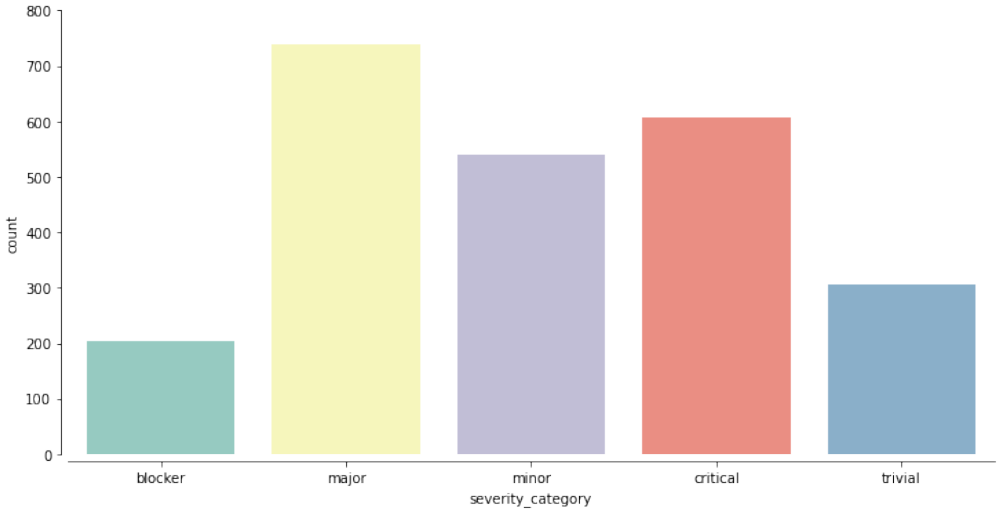


Figure 1: Target label distribution on the Mozilla bug reports dataset.

## 2.2 Evaluation Metrics

The specific metrics I’ll use for assessing prediction performance for evaluating the accuracy of classification algorithms will be precision, recall, and F-measure, described as follows[6]:

- **Recall**. The recall is the number of True Positives (TP) divided by the number of True Positives (TP) and of False Negatives (FN), where the TP and FN values are derived from the confusion matrix. A low recall indicates many false negatives.
- **Precision**. Precision is the number of True Positives (TP) divided by the number of True Positives and False Positives (FP). A low precision can also indicate many false positives.
- **F-measure**. F-measure conveys the balance between precision and recall and can be calculated as their harmonic mean.

The metrics above especially useful when the target classes are unbalanced, which often happens in many settings, such as bug severity prediction. Also, these metrics enable the comparison between my prediction model and benchmark models.

## 2.3 Benchmark model

The benchmark model that will be used in my project will be based on the literature. Table 2 provides a quantitative view of the performance obtained in the experimental setting, showing the algorithms which presented the best performances in some papers.

| Ref. | Projects | Algorithms | Hyperparameters | Classes | Metrics | | | Terms |
|---|---|---|---|---|---|---|---|---|
| | | | | | AUC | Accuracy | F-Measure | |
| [17] | Mozilla | Multinomial Näive Bayes | | 2 | 0.849 | | | 100 |
| [18] | Mozilla | KNN | K = 1 | 7 | | 0.750 | | 40 |
| [15] | Mozilla | Random Forest | | 2 | | 0.823 | 0.432 | |
| [13] | Mozilla | Näive Bayes | | 6 | | 0.916 | | |
| [12] | Mozilla | Näive Bayes | | 2 | 0.924 | 0.880 | 0.797 | 750 |
| [16] | Mozilla | Bagging Ensemble | | 2 | | | 0.482 | |
| [11] | Mozilla | Bagging Ensemble | | 4 | | 0.812 | | |
| [9] | Mozilla | Random Forest | | 2 | | 0.764 | | 59 |
| [3] | Mozilla | Multinomial Näive Bayes | | 2 | | | 0.820 | 79,000 |
| [4] | Mozilla | Multinomial Näive Bayes | | 2 | | | 0.820 | |
| [5] | Mozilla | Multinomial Näive Bayes | | 2 | | | 0.790 | |

Table 2: The best ML algorithms performance of papers in the literature.

# 3 Project Design

Figure 2 shows the high-level architecture of the bug severity predictor application. The architecture will consist of two modules: front-end and back-end modules. The front-end module will run in a web browser, and it will be in charge of receiving and displaying data from or to the user. The back-end module will run in a server and be responsible for cleaning, breaking into tokens, and weighting each term of the bug description text extracted from a BTS system. Moreover, the back-end module will predict the bug severity level, as requested of the front-end, based on the bug description and the predicting model previously saved. The front-end module will communicate with the back-end throughout the REST protocol.

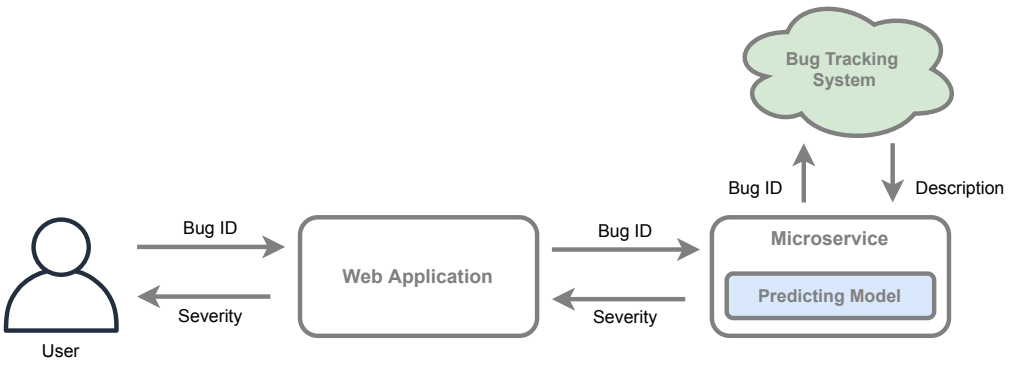


Figure 2: High-level architectural design.

The predicting model will be train and test using features extracted using the Pre-training of Deep Bidirectional Transformers for Language Understanding (BERT)[2].

Figure 3 shows the main activities of the project to implement the proposed architecture above, which are described as the following:

1. **Clean, tokenize and weight the description feature:** This activity will clean and tokenize the description feature of the Mozilla dataset. Next, it will weight the tokens of each cleaned description. To do that, I'll use the NLTK Python library and BERT deep learning architecture.
2. **Split the dataset:** This activity will split the processed dataset in training (75%), and testing sets (25%). To do that, I'll use the Scikit-learn Python library.

3. **Train and tune the model:** This activity will train and tune the predicting model. To train, I'll use the XGBoost algorithm from the Scikit-learn library, and, to tune, I'll use the grid-search with 5-fold cross-validation.
4. **Calculate and evaluate training metrics:** This activity will calculate and evaluate training metrics. To calculate metrics I'll the methods from the Scikit-learn library and, to evaluate, I'll plot the charts using from Python Matplotlib library.
5. **Test and evaluate the model:** This activity will test and evaluate the predicting model. To test, I'll use methods from the Scikit-learn library, and, to evaluate, I'll plot charts using the Python Matplotlib library.
6. **Calculate and evaluate testing metrics:** This activity will calculate and evaluate testing metrics. To calculate metrics I'll the methods from Scikit-learn library and, to evaluate, I'll plot charts using from Python Matplotlib library.
7. **Deploy in production:** This activity will deploy a model in a web application in the cloud. To develop the application, I'll use the Flask Python library for the back-end, and HTML, Javascript, and CSS for the front-end.

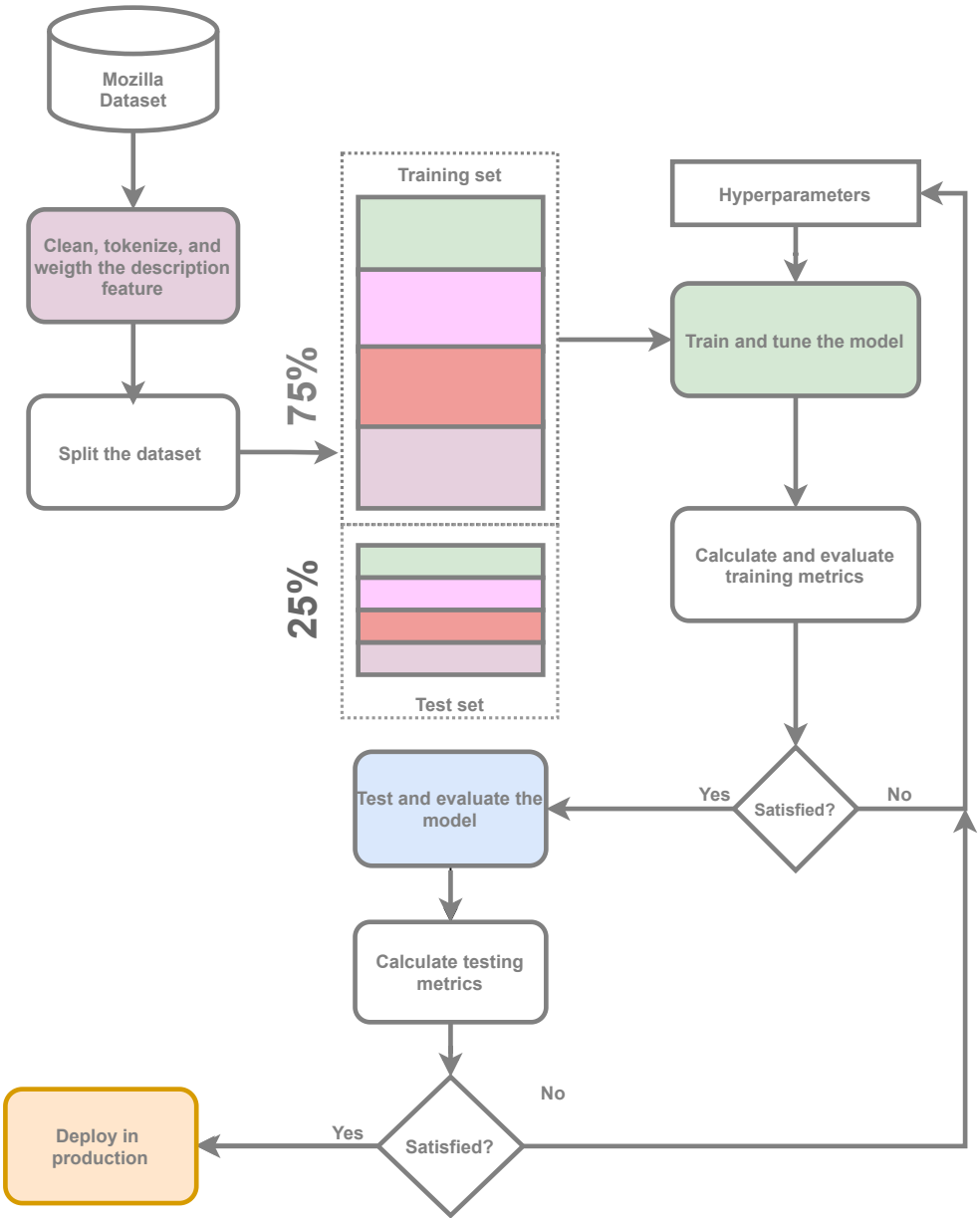


Figure 3: Machine learning workflow of the projetc.

# References


[1] Yguaratã Cerqueira Cavalcanti et al. “Challenges and opportunities for software change request repositories: a systematic mapping study”. In: *Journal of Software: Evolution and Process* 26.7 (July 2014), pp. 620–653. ISSN: 20477473. DOI: 10.1002/smr.1639.

[2] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://www.aclweb.org/anthology/N19-1423.

[3] Kwanghue Jin et al. “Bug Severity Prediction by Classifying Normal Bugs with Text and Meta-Field Information”. In: *Advanced Science and Technology Letters* 129 (2016), pp. 19–24. DOI: http://dx.doi.org/10.14257/astl.2016.129.05.

[4] Kwanghue Jin et al. “Improving Predictions about Bug Severity by Utilizing Bugs Classified as Normal”. In: *Contemporary Engineering Science* 9 (2016), pp. 933–942. DOI: http://dx.doi.org/10.12988/ces.2016.6695.

[5] Kwanghue Jin et al. “Utilizing Feature based Classification and Textual Information of Bug reports for Severity Prediction”. In: *Information* 19.2 (Feb. 2016), pp. 651–659.

[6] Max Kuhn and Kjell Johnson. *Applied predictive modeling*. New York, NY: Springer, 2013.

[7] Ahmed Lamkanfi et al. “Predicting the severity of a reported bug”. In: *Proceedings - International Conference on Software Engineering* (2010), pp. 1–10. ISSN: 02705257. DOI: 10.1109/MSR.2010.5463284.

[8] Ahmed Lamkanfi et al. “Comparing mining algorithms for predicting the severity of a reported bug”. In: *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR* (2011), pp. 249–258.

[9] A. F. Otoom et al. “Severity prediction of software bugs”. In: *2016 7th International Conference on Information and Communication Systems (ICICS)*. Apr. 2016, pp. 92–95. DOI: 10.1109/IACS.2016.7476092.

[10] Procrastinator. *How to Improve Class Imbalance using Class Weights in Machine Learning*. URL: https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/. (accessed: 16.01.2021).

[11] 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. DOI: 10.1109/CIMCA.2016.8053276.

[12] 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. DOI: 10.1109/SEAA.2014.51.

[13] Meera Sharma et al. “Multiattribute Based Machine Learning Models for Severity Prediction in Cross Project Context”. In: *Computational Science and Its Applications – ICCSA 2014*. Ed. by Beniamino Murgante et al. Cham: Springer International Publishing, 2014, pp. 227–241. ISBN: 978-3-319-09156-3. DOI: 10.1007/978-3-319-09156-3_17.

[14] 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. DOI: 10.1109/WCRE.2012.31.

[15] Harold Valdivia Garcia and Emad Shihab. “Characterizing and Predicting Blocking Bugs in Open Source Projects”. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: ACM, 2014, pp. 72–81. ISBN: 978-1-4503-2863-0. DOI: 10.1145/2597073.2597099.

[16] Xin Xia et al. “ELBlocker: Predicting blocking bugs with ensemble imbalance learning”. In: *Information and Software Technology* 61 (2015), pp. 93–106. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2014.12.006.

[17] C. Z. Yang et al. “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. DOI: 10.1109/APSEC.2012.144.

[18] 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. DOI: 10.1109/COMPSAC.2014.16.

[19] Geunseok Yang et al. “Analyzing Emotion Words to Predict Severity of Software Bugs: A Case Study of Open Source Projects”. In: *Proceedings of the Symposium on Applied Computing*. SAC ’17. Marrakech, Morocco: ACM, 2017, pp. 1280–1287. ISBN: 978-1-4503-4486-9. DOI: 10.1145/3019612.3019788.