Reproduction of "Predicting the Severity of a Reported Bug"

Andrina Vincenz
andrina.vincenz@uzh.ch
University of Zurich
Zurich, Switzerland

David Stalder
david.stalder@uzh.ch
University of Zurich
Zurich, Switzerland

ABSTRACT

Background. Bug fixing is a fundamental part of software maintenance. Bugs have different levels of severity according to their threat to the system and urgency to fixing.

Aim. The aim is to have models which are able to identify the severity of bug reports.

Method. After identifying the system, bug reports will be extracted from bugzilla including the severity of the bug. This data is used to train two classifiers: SGDClassifier and Multinomial Naïve Bayes. The following measurements are used to identify the quality of the model: precision, recall and f-measures.

Conclusion. This study will show if it is possible to generate models, which can do the classification of bug reports automatically with certainty. This can help software engineers in practice to classify their bug reports and focus on the most urgent.

1 INTRODUCTION

Bug reports represent a fundamental part of software maintenance by documenting the malfunction of a software part. A large number of bug reports by users and developers are collected on a daily basis in bug tracking systems like Bugzilla. These reports are then manually prioritized according to their severity and classified into 6 groups, ranging from a request to enhancement to a critical level resulting in crashes, loss of data or severe memory leak. The level of severity indicates the impact of the bug on the successful execution of a software system.

However, the manual assignment of the severity level to a bug report takes time and resources. In combination with the increase of daily bug reports the classification introduces a set of challenges such as misclassification of severity or the increase of cost. The ability of the automated severity prediction of bug reports by a tool reduces errors in misclassification and manual work as well as enhance the software quality.

In order to automate the severity predication of bug reports, different classification and information retrieval techniques based on textual descriptions have been used previously. Techniques range from Nearest Neighbours [6] [1] [5], Naïve Bayes [1] [5] to Support Vector Machines [4] [1] [5]. While some work focuses on automatically analyzing the text of past bug reports and the recommendation of severity labels [6] [1] [5] other work creates a classification-based approach to create a bug priority recommender [4].

For this project, we intend to conduct a reproduction of the paper "Predicting the severity of a reported bug" [2] by Lamkanfy et al. It was originally written in 2010, which we consider as too old for such a current topic.

The research goal of this paper was to test if it is possible to predict the severity of a reported bug by analysing its textual description by using a text mining algorithm. They came up with subsidiary research questions which need to be included to find an explanation of their research goal.

- Concerning the length of a bug report, the research question was: "Which (text) fields in the bug reports serve as the best prediction basis? The one-line summary which briefly focusses on the problem or the longer full description which includes more detail?" [2]
- Concerning the scope of the training period, their research question was: "How many samples must be collected before one can make a reliable predictor?" [2]
- Concerning the components their research question was: "Is it better to have a specialized predictor for each component of the software system, or can we combine bug reports over different software components (the so-called "cross-component" approach)?" [2]

At the end of the paper they already analysed those research questions. However, we will base our work on the findings which were made by them as follows:

- *Short vs. Long*: They found that there is no difference in the prediction basis between the one-line summary and the full description [2]. To have a smaller set of samples and to make the procedure as efficient as possible, we decided to use only the one-line summary.
- *Training Period*: They found that there is nearly no difference in the reliability of the predictor by taking 500, 1000 or 2000 samples [2]. To be sure that our findings will be as precise as possible, we decided to take 2000 samples.
- *Per Component vs. Cross Component*: They found that using the cross-component approach the performance improves with an increasing number of samples [2]. Therefore we decided to use the cross-component approach, since we are already using 2000 samples.

The structure of the paper is as follows. In the second section "Methodology" we focus on how we conducted our research and explain every step of the work. In the third section "Results" we present our findings, in the fourth section "Discussion and Limitations" we discuss and interpret the findings and name the limitations of our research.

We conducted the procedure as follows: First we downloaded and processed the data such that we could use it for the further procedure. For each of the six severity types we chose 2000 entries for the training list and 200 entries for the testing list. In the next step we trained the two models (Naïve Bayes and SGD). For measuring purposes we used the scores precision, recall and f1, the parameter optimization was achieved using grid search. In the last step the classifiers were applied on the test set. Our experimental results for the Naïve Bayes were 0.402 for all the scores (precision, recall and f1), for the SGD the experimental results for all the scores were

0.405. Finally, the severity types "trivial" and "critical" had with both methods an accuracy of over 60% and all the other types had an accuracy of between 30-45%.

2 METHODOLOGY

In a first step we downloaded the necessary data from Bugzilla tracking system with bugs regarding the case Firefox. In our approach only one case was selected. Since we choose the cross-component approach all components were selected. To make the analysis clearer, the only resolution type chosen is fixed. This is because fixed bugs were worked on and therefore the label more certain. There are eight types of severities: "blocker", "critical", "major", "normal", "minor", "trivial", "enhancement" and "N/A". The categories "enhancement" and "N/A" were dropped, "N/A" since this label has no meaning regarding the problem statement and "enhancement" because this class is imbalanced with only 45 entries in total as well as these are user requests for features. In contrast to the original paper we decided to run the models on all severities as labels to find out whether we get a similar accuracy or worse. To make the training and test step more coherent, the test and training set were downloaded separately. The test set contains each 2000 entries for each severity resulting in 12000 entries. For the test set we have 200 entries per severity with 1200 entries in total.

After downloading the data, our approach consists of the following steps.

(1) *Preprocess bug reports*: Afterwards the data was imported into a Jupyter Notebook. The data was further preprocessed by dropping rows with missing values or labels resulting in a training set with 11962 entries and a test set with 1999 entries. Further the punctuation and special characters were removed from the bug summary. For the preprocessing of the bug summary a pipeline was used transforming the words into ngramms ranging from (1,1) to (1,3). Further the features were transformed into the tfidf measure reflecting the importance of a word in a document. The label encoding was done with the method LabelEncoder() from sklearn.
(2) *Training the classifiers*: With the training set two different classifiers were trained. First the NB and SGD classifier were trained without any parameter optimization. To measure the accuracy of each model we used the following scores: precision, recall and f-measures. The calculation of the measurements was done via cross validation. Parameter optimization for both classifiers was achieved with Grid Search. For Naïve Bayes only ngramm optimization was performed. For the SGD classifier the following parameters additionally were optimized: loss function, regularization, and early stopping. The grid search resulted in a best model for each classifier.
(3) *Applying classifiers on test set* To calculate the accuracy of each classifier the best model was run with the test set and the results visualized with a normalized confusion matrix.

The whole procedure can be found on our GitHub repository [7]

3 RESULTS

Table 1 shows the precision, the recall, and the f1-score of the two selected classifiers using the data samples from Mozilla.

The results of the two different classifiers are nearly the same, whereas the precision, the recall and the f1-score are nearly identical and varying between 0.40-0.41.

Table 1: Scores of the two Classifiers

Classifier	Precision	Recall	f1
Naïve Bayes	0.402	0.402	0.402
SGD	0.405	0.405	0.405

In the next step we used a grid-search for finding the best model of both classifiers. Figure 1 is the normalized relative confusion matrix of the best Naïve Bayes model which has an accuracy score of 0.466, Figure 2 is the normalized relative confusion matrix of the best SGD model which has an accuracy score of 0.455.

Figure 1: Relative Confusion Matrix of Naïve Bayes Model

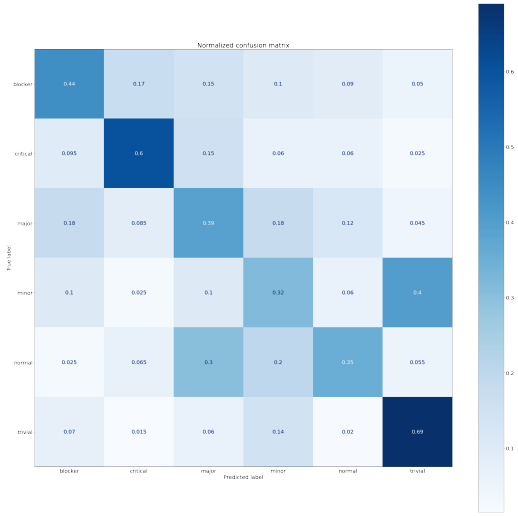


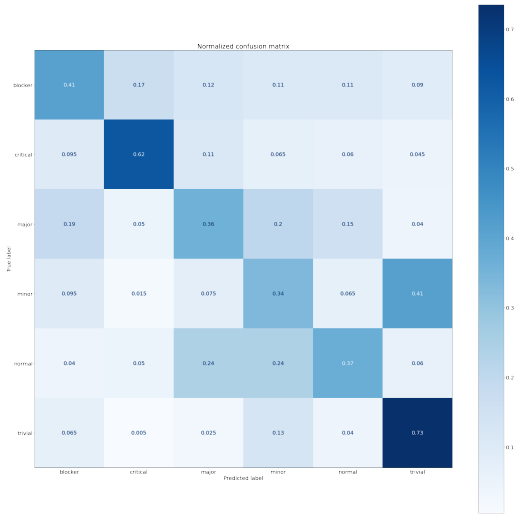
Figure 1 shows that in the Naïve Bayes model the label "blocker" had an accuracy of 0.44, "critical" 0.6, "major" 0.39, "minor" 0.32, "normal" 0.35, and "trivial" 0.69.

Figure 2 shows that in the SGD model the label "blocker" had an accuracy of 0.41, "critical" 0.62, "major" 0.36, "minor" 0.34, "normal" 0.37, and "trivial" 0.73.

4 DISCUSSION AND LIMITATIONS

This approach shows that classifying bugs according to their severity with linear classifiers only works to a certain point. Only two labels were classified with a higher accuracy than 50%, namely "trivial" and "critical". The reason for this could be that the bug summaries over all components have too similar words to be able to classify them correctly. In this case there is no better solution available with linear models except for larger ngramms. Is this not the case, larger test and trainings files could lead to a higher accuracy for this approach. Compared to the paper [2] with the same amount of test data for the non-component approach it shows that classifying the bug severity further into two categories leads to a higher accuracy. Training another linear classifier leads to no improvement in accuracy and to similar measurements.

Figure 2: Relative Confusion Matrix of SGD Model



As already mentioned in the section "Results", we observed that in both models the values for precision, recall and f1-score are nearly identical. To be able to find possible explanations, we first need to have a look at the formulae of each score [3]. In the following formulae, TP means "true positive", FP means "false positive", and FN means "false negative".

Precision:

$$p = \frac{TP}{TP + FP} \tag{1}$$

Recall:

$$r = \frac{TP}{TP + FN} = \frac{TP}{TP + FP} \tag{2}$$

f1:

$$f1 = \frac{(1 + 1^2)TP}{(1 + 1^2)TP + 1^2FN + FP} = \frac{2TP}{2TP + FP + FN} = \frac{TP}{TP + FP} \tag{3}$$

Since all numbers are nearly the same, in our case we seem to have fp=fn (see in Equations 1, 2, and 3), i.e. the number of false positives must be equal the number of false negatives which makes all equations equal. This can be interpreted as good, since our data set had an equal amount of each label.

REFERENCES

[1] Q.D. Soetens T. Verdonck A. Lamkanfi, S. Demeyer. 2011. Comparing mining algorithms for predicting the severity of a reported bug. *2011 15th European Conference on Software Maintenance and Reengineering* (2011), 249–258. https://doi.org/10.1109/CSMR.2011.31

[2] S. Giger B. Goethals A. Lamkanfi, S. Demeyer. 2010. Predicting the severity of a reported bug. *7th Working Conference on Mining Software Repositories, Cape Town, South Africa, 2 May 2010 - 3 May 2010* (2010), 1–10. https://doi.org/10.1109/MSR.2010.5463284

[3] E. Gaussier C. Goutte. 2005. A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation. *Advances in Information Retrieval* (2005), 345–359. https://doi.org/10.1007/978-3-540-31865-1_25

[4] O. Maqbool J. Kanwal. 2010. Managing open bug repositories through bug report prioritization using SVMs. *Proceedings of the 4th international conference on open-source systems and technologies* (2010), 1–7. https://doi.org/ICOSST

[5] V.B. Singh K.K. Chaturvedi. 2012. Determining bug severity using machine learning techniques. *CSI sixth international conference on software engineering* (2012), 1–6. https://doi.org/10.1109/CONSEG.2012.6349519

[6] Sun C. Tian Y., Lo D. 2013. DRONE: predicting priority of reported bugs by multi-factor analysis. *2013 IEEE International Conference on Software Maintenance* (2013), 200–209. https://doi.org/10.1109/ICSM.2013.31

[7] Andrina Vincenz and David Stalder. 2021. *Project for Seminar: Data Science in for Software Engineering.* Retrieved February 08, 2021 from https://github.com/dastal/Data_Science_for_Software_Engineering/tree/master/Project