2016 42th Euromicro Conference on Software Engineering and Advanced Applications

Automated Bug Triaging in an Industrial Context

Václav Dedík
Faculty of Informatics
Masaryk University, Brno, Czech Republic
dedik@mail.muni.cz

Bruno Rossi
Faculty of Informatics
Masaryk University, Brno, Czech Republic
brossi@mail.muni.cz

***Abstract*—There is an increasing need to introduce some form of automation within the bug triaging process, so that no time is wasted on the initial assignment of issues. However, there is a gap in current research, as most of the studies deal with open source projects, ignoring the industrial context and needs.**
In this paper, we report our experience in dealing with the automation of the bug triaging process within a research-industry cooperation. After reporting the requirements and needs that were set within the industrial project, we compare the analysis results with those from an open source project used frequently in related research (Firefox). In spite of the fact that the projects have different size and development process, the data distributions are similar and the best models as well. We found out that more easily configurable models (such as SVM+TF–IDF) are preferred, and that top-x recommendations, number of issues per developers, and online learning can all be relevant factors when dealing with an industrial collaboration.

***Keywords*-Software Bug Triaging; Bug Reports; Bug Assignment; Machine Learning; Text Classification; Industrial Scale;**

I. Introduction

In this paper, we deal with bug triaging, that is the process of assigning new issues or tickets to developers that could better handle them. Most of the papers in the area of automated bug triaging deal with Open Source Software (OSS) projects (e.g. [1], [2]). In the current paper we take a point of view that has not been investigated often, that is the different needs in terms of approaches needed in case of research-industry collaboration. In fact, OSS has a distributed development model [3] that implies a different triaging process but also different results during the application of an automated triaging process. For this reason, we set-up the main research question: *what are the differences in bug triaging automation when considering machine learning models between an OSS project and a company-based proprietary project?* From this main research question, throughout the paper we provide several sub-questions that can give more insights on different aspects.

To answer the main research question, we conducted an experimentation within a software company from the Czech Republic, running a SCRUM-based development process centered around several teams and a JIRA issue tracker that provided the central part of the triaging process. At the time of starting the experimentation, the tickets were left unassigned until one responsible would start the triaging process. We dicuss in the paper the differences — when implementing an automated triaging system — that were detected between such proprietary-based project and data from Mozilla Firefox.

II. Related Works

Over the years there have been many attempts to automate bug triaging. The first effort was made by Čubranić and Murphy [1] using a text categorization approach with a Naïve Bayes classifier on Eclipse data. Their dataset consisted of 15,670 bug reports with 162 classes (developers). They achieved about 30% accuracy.

Anvik et al. [2] used Support Vector Machines (SVM) on Eclipse, Firefox and GCC data. They achieved precision of 64% and 58% on Firefox and Eclipse data, however, only 6% on GCC data. As for recall, only 2%, 7% and 0.3%: results were achieved on Firefox, Eclipse and GCC data.

Ahsan et al. [4] used an SVM classifier on Mozilla data. They reached 44.4% classification accuracy, 30% precision and 28% recall using SVM with LSI.

An extensive study was done by Alenezi et al. [5], using a Naïve Bayes classifier. The best results were achieved using χ^2 with precision values of 38%, 50%, 50% and 50% and recall values of 30%, 35%, 21% and 46% on Eclipse-SWI, Eclipse-UI, NetBeans and Maemo projects respectively.

Xia et al. [6] employed an algorithm named DevRec based on multi-label k-nearest neighbor classifier (ML–kNN) and topic modeling using Latent Dirichlet Allocation (LDA). For top-5 recommendation, the recall values were 56%, 48%, 56%, 71%, 80% and the precision values 25%, 21%, 25%, 32%, 25% for GCC, OpenOffice, Mozilla, NetBeans and Eclipse.

Automated bug triaging in industrial projects has not been discussed often (in [7], only *two* out of 25 studies reviewed). Hu et al. [8] provided an evaluation of automatic bug triaging considering a component-developer-bug network on two industrial projects (1,008/686 bug reports, 19/11 developers) and three OSS projects (Eclipse, Mozilla, Netbeans). They overall noticed that the proposed approach performs worse on the industrial projects. Jonsson et al. [7] represent one recent study to underline the missing availability of automatic triaging studies within the industrial context. Using an ensemble learner, authors collect 50,000 bug reports from five projects out of two companies, showing that the system can scale well with large amounts of training sets. Main findings are that SVM models perform well in industrial context and that models performance drops when projects team size increases.

III. Experimental Evaluation

The process of triaging automation begins by retrieving a dataset from a bug tracking system (Fig. 1). The next step

2376-9505/16 $31.00 © 2016 IEEE
DOI 10.1109/SEAA.2016.20

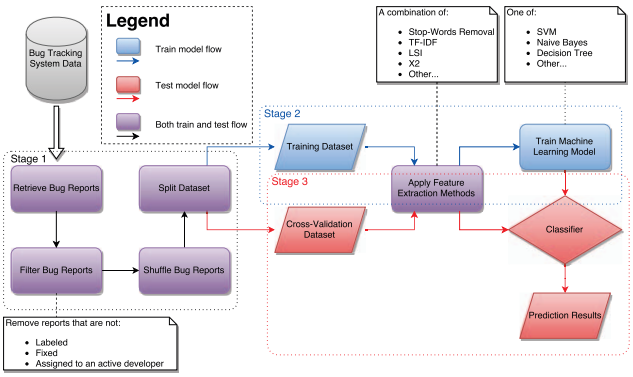

Fig. 1. Data analysis process: from bug tracking data to the prediction results.

is to filter unwanted bug reports from the dataset. All tickets that are unassigned, not fixed or not resolved are removed. There can be also bug reports that are assigned to a universal computer-generated user (e.g. `nobody@mozilla.org`) — we remove these reports as well. Bug reports resolved by developers that have not fixed sufficient amount of reports in the past (e.g. 30) are also removed. Another step is to shuffle the bug reports randomly. This is done to achieve more accurate performance results with the cross-validation (CV) set. The process continues by splitting the resulting dataset into two sets—a cross-validation set and a training set. The CV set contains 30% of the bugs while the training set contains the remaining 70%.

The second stage is to train the machine learning model (e.g. SVM, NB). First, however, it is necessary to run the training dataset through a feature extraction step. This step applies techniques that improve the performance of a machine learning model, e.g. stop-words removal, TF–IDF etc...

The last stage uses the trained machine learning model to generate prediction results that can be used, for example, to compute various performance metrics. The first step is to (again) apply feature extraction techniques on the CV dataset and then use the trained classifier to predict results.

One of the most prominent models for text classification is known as Support Vector Machine (SVM). Instead of trying to find a function that fits training data as precisely as possible by optimizing the distance from samples to the function, the SVM algorithm attempts to find a linear separator of positive and negative samples by optimizing the margin of the decision boundary (*hyperplane*) [9]. The samples that limit the margin of the decision boundary on both sides are called support vectors. However, positive and negative samples can be mixed up together so much that it is impossible to construct a decision boundary. This is why the strictness of the optimization algorithm can be relaxed by tuning the regularization parameter C [9].

In many applications, the linear classification is not enough to warrant a good decision boundary and performance. Using kernel functions, we can effectively transform the decision boundary into polynomial, Gaussian or sigmoid function. By using a technique called *kernel trick*, this can be done without increasing the time complexity of the learning algorithm [9].

To evaluate the performance of the models, we considered three metrics — accuracy, precision and recall[1].

1) Accuracy: This metric measures the overall fraction of correctly predicted assignees out of all predictions [9].

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{1}$$

2) Macro-Averaged Precision: Precision measures the fraction of the correctly predicted assignees as positive out of the assignees either correctly or incorrectly predicted as positive. A value is computed separately for every class (assignee) and the result is computed as the mean of all these values.

$$Precision_{macro} = \frac{1}{q}\sum_{\lambda=1}^{q}\frac{tp_\lambda}{tp_\lambda + fp_\lambda} \tag{2}$$

3) Macro-Averaged Recall: Similar to precision, except recall measures the fraction of the correctly predicted assignees as positive out of the assignees either correctly predicted as positive or incorrectly predicted as *negative*. Again, the macro-averaged variant is computed as the mean of all recall values for every class (assignee).

$$Recall_{macro} = \frac{1}{q}\sum_{\lambda=1}^{q}\frac{tp_\lambda}{tp_\lambda + fn_\lambda} \tag{3}$$

A. Dataset description

The Proprietary dataset was collected within the context of the collaboration with a software company from the Czech Republic. The development process is run with a SCRUM-based process centered around several teams and a JIRA issue tracker that collects all the issues. At the time of starting the experimentation, the tickets were left unassigned until one responsible would start the triaging process. The mined dataset contains $2,926$ bug reports created between 2012-11-23 and 2015-10-16. Only bug reports that were resolved with assigned developers were considered. There are 110 developers in this dataset. Only $2,424$ bugs assigned to 35 developers were retained after removal of developers with less than 30 fixed bugs (Fig. 2).

The OSS dataset was mined from the Mozilla repository from project Firefox[2]. We downloaded all bugs that are in status `RESOLVED` with resolution `FIXED` and were created in year 2010 or later. We also removed bugs with field `assigned_to` set to `nobody@mozilla.org` as those tickets were not assigned to an actual developer. In total, we were able to retrieve $9,141$ bugs. To get a better comparison with the other datasets, we only use $3,000$ data points for training and cross-validation that were created between 2010-01-01 and 2012-07-10. This dataset contains 343 labels (developers). Finally, we remove developers who did not fix at least 30 bugs, yielding $1,810$ bugs with 20 developers (Fig. 3).

[1]for both precision and recall, we considered the macro-averaged variants.
[2]https://bugzilla.mozilla.org

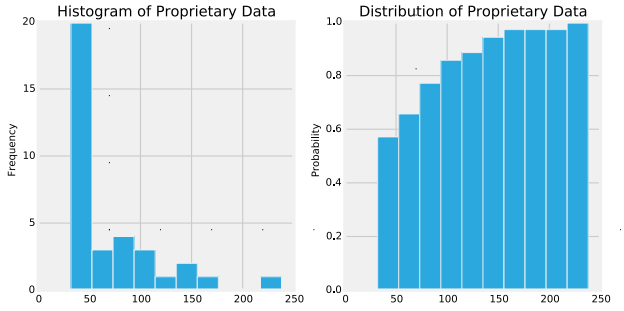

Fig. 2. Proprietary project: distribution of tickets per single developer.

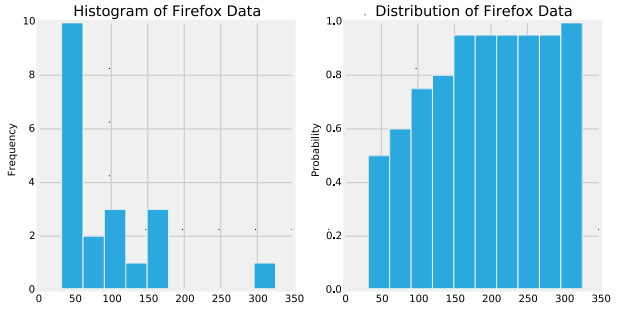

Fig. 3. Firefox project: distribution of tickets per single developer.

B. Experimental Questions

For the analysis, we structured the experimental part linking the original goal to several questions:

- *Q1/Q2. Are the two datasets coming from the same distribution? Do the samples have the same population mean?* This would allow to determine how similar the two datasets are in terms of issues distribution;
- *Q3. Do classification models bring benefits over a baseline classifier?* This allows us to review the relevance of the provided models;
- *Q4. Which classification model brings the best results for both datasets in terms of accuracy, precision and recall?* This allows us to look into differences of performance of the models taking into account the two datasets;
- *Q5. Taking into account the best identified models, how do they perform considering different number of issues per developers?* Companies want to understand how much time needs to pass until one developer can be included in the prediction process;
- *Q6. Taking into account the best identified models, what is the performance of the models considering a higher number of recommendations?* One of the requirements of companies is to provide a ranking list of assignee that could be useful in case of re-assignments of tickets;

C. Dataset distributions (Q1/Q2)

We use the two-sided alternative of the two-sample chi-square test to evaluate the null hypothesis that two samples are from the same distribution. The test with Firefox and Proprietary samples returns a p-value equal to 0.8648 so we fail to reject the null hypothesis that the two samples are from the same distribution (5% significance).

After we tested the hypothesis that the used datasets come from the same distribution, we test the null hypothesis that the samples have the same population mean—allowing us to further learn how statistically similar the datasets are, looking at the variances of the samples. We use the two-sided alternative of the standard independent two-sample t-test to test the null hypothesis that the samples have the same population mean, and the two-sided alternative of the Levene test to test whether the samples have equal population variance. Testing data from Firefox and Proprietary datasets, the Levene test yields p-value equal to 0.2730, as p-value > 0.05, we fail to reject the null hypothesis (5% significance). To test the population mean of the two samples, we therefore have to use the standard independent t-test. The standard independent t-test results in p-value of 0.1954, which means we fail to reject the null hypothesis of equal population mean of the two samples (5% significance).

Chi-square and t-test imply there is little difference between proprietary and Firefox dataset in terms of data distribution.

D. Comparison with the Baseline Model (Q3)

A first step on the evaluation of the models was to compare against a baseline model. We defined such model as a classifier that would assign a bug report to the developer with the highest number of reports. While the accuracy of this model is relatively high (18%), the precision and recall values are much lower (1% and 5%) on the Firefox data (Fig. 4).

There is also an increase in performance after all stop-words were removed from the feature vector of Firefox data. The performance of the classifier slightly increased on all models. Accuracy increased by 3% on SVM. The precision value of the SVM model decreased by 1% but increased by 6% with Naïve Bayes. Finally, Recall values of SVM and Naïve Bayes increased by 4% and 2% respectively. We therefore conclude that the performance boost of stop-words removal is significant enough to warrant better results, which matches the conclusion of Čubranić and Murphy [1].

The initial models bring benefits over the baseline classifier.

E. Comparison of the Models (Q4)

Based on related work, we compared two models: Naïve Bayes (see previous work from one of the authors, [10]), and SVM. Parameters of all models were optimized by grid search.

SVM+TF–IDF offers the best performance of 53% accuracy, 59% precision and 47% recall. The same model with LSI also shows quite good performance and the Naïve Bayes model with χ^2 and TF–IDF performs quite well as far as precision is concerned. Both SVM and Naïve Bayes models exhibit rather wide spreads between precision and recall values

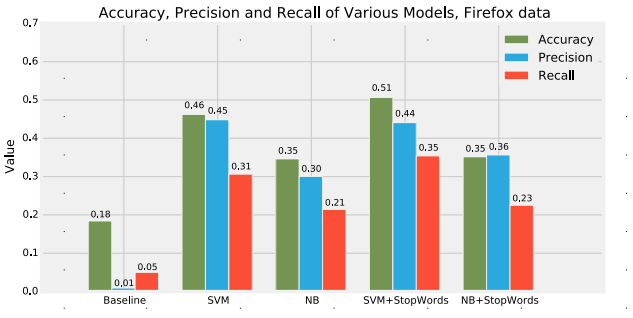

Fig. 4. Firefox project: baseline model and stop-words removal.

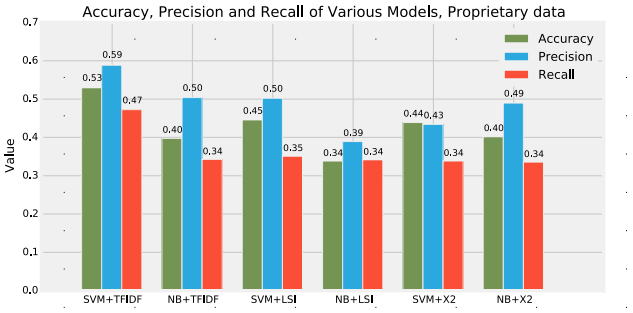

Fig. 5. Proprietary project: models comparison.

(Fig. 5), which could be an indication of higher variance of the proprietary data (Fig. 2 and 3).

SVM+TF–IDF achieves the best performance with accuracy of 57%. Its precision and recall also outperforms all the other approaches with values of 51% and 45% (Fig. 6). The comparison shows that SVM+TF–IDF performs best on all datasets. It also generalizes very well, because there was no need to readjust the parameters. The disadvantage of the model is that it is the most computationally complex one, because SVM is the slowest as there are many classes and features. This can be at least partially dealt with by using χ^2 feature extraction in conjunction with TF-IDF while sacrificing some of the classification performance.

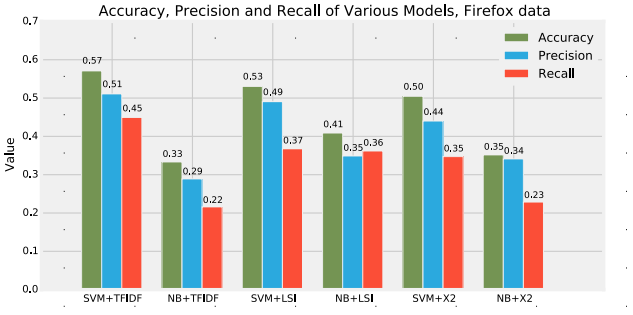

Fig. 6. Firefox project: models comparison.

SVM+TF–IDF+Stop Words Removal is the best model for both proprietary company-based and Firefox data.

F. Number of Issues per Developer (Q5)

We compared the two datasets using the SVM+TF–IDF weighting by computing their performance (accuracy, precision and recall) for six different settings of the minimum *issues per developers* (1, 3, 5, 10, 20, 30) requirement (Fig. 7). The accuracy of the proprietary dataset (53%) is a bit lower than the accuracy of the open-source dataset (54%) when minimum issues per developer equals 30. Interesting fact is that the initial behaviour is quite different with much higher performance from the proprietary dataset (42%) than the open-source dataset (29%).

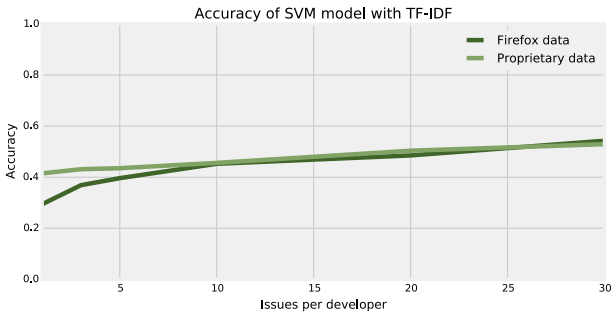

Fig. 7. Comparison of accuracy of the SVM model.

The precision value of the proprietary dataset (Fig. 8) is eventually higher (59%) than the precision value of the open-source dataset (54%). In the recall case, the right-most case (minimum number of issues per developer requirement equal to 30) shows the performance of the proprietary dataset slightly lower (47%) than that of the open-source dataset (50%). Also in this case, the performance for minimum number of issues per developer equal to *one* is much higher for the proprietary dataset (14% vs 3%).

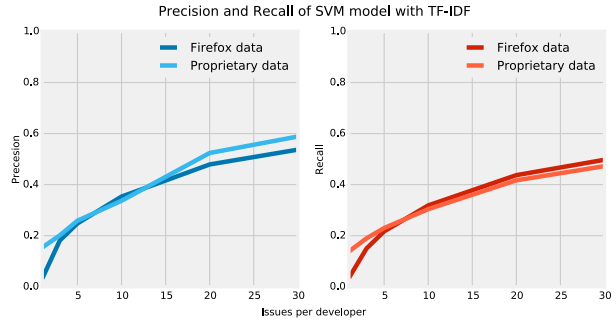

Fig. 8. Comparison of precision of the SVM model.

The performance of the proprietary dataset is generally quite similar to that of the open-source dataset. An interesting conclusion is that the spread between precision and recall is much higher for the proprietary dataset. This possibly implies

higher variance of the dataset. All our results show significant difference in performance for minimum number of issues per developers equal to *one*. The higher performance for the proprietary dataset in this regard can be probably explained by the fact that open-source bug repositories are open to anyone.

One-time assignees can impact negatively on the performance when considering different nr. of issues per developer.

G. Performance for Higher Recommendations Number (Q6)

We examined the performance of the models with different number of recommendations. We show the performance of the SVM model with TF-IDF weighting trained on the proprietary and Firefox datasets for number of recommendations from *one* to *ten* (Fig. 9). The accuracy increases with the number of recommendations, which is expected as the more recommendations, the higher the chance of a hit (i.e. the chance that the list of predictions contains the correct assignee).

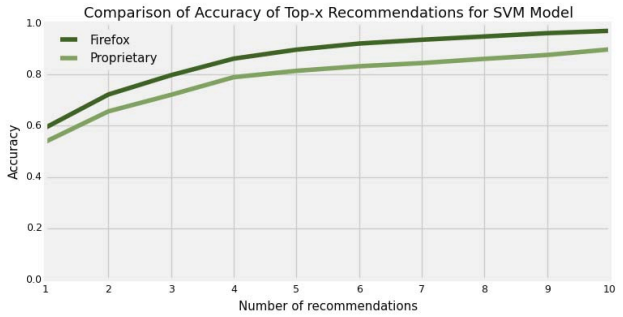

Fig. 9. Comparison of accuracy of top-x recommendations for SVM model.

It is apparent from the plot that the highest performance boost happens when the number of recommendations changes from *one* to *two* both for the proprietary dataset (54% vs 66%) and the Firefox dataset (59% vs 72%). The accuracy of the Firefox dataset is equal to 90% for 5 recommendations and 81% for proprietary dataset. When the number of recommendations is 10, the performance is 97% and 90% for the Firefox and proprietary data respectively.

Similar behaviour in changes of accuracy when considering the variation of the number of recommended developers.

IV. Discussion & Conclusion

In the current paper, we evaluated the usage of an automated bug triaging process within a Czech Republic-based company to look at similarities and differences to the usually analyzed open source projects. We found that the best classification model is SVM with TF–IDF— achieving an accuracy of 53%, precision of 59% and recall of 47%— not dissimilar from the Firefox dataset: 57%, 51% and 45% respectively. One appreciated advantage of such model within the company is that it does not require large parameters optimization. From the point of view of the many aspects we analyzed, the distribution of the datasets seems to be very similar (t-test and chi-square test). Subsequent evaluation of performance of the datasets using SVM and TF-IDF supports this conclusion further. Unfortunately, this conclusion does not apply for all feature extraction methods, our results show that it is necessary to choose different parameters for both LSI and χ^2 techniques. In term of performance, however, there is a significant difference when we omit filtering of developers that fix few bugs. The most likely explanation is that the company-based proprietary dataset contains less developers that fixed only a few bugs, as the proprietary bug repository is not public.

Comparing the performance of the models with the related work can be problematic due to different processes of analysis, datasets and metrics used. Overall, the results on the Firefox project can be considered comparable with our results. There can be mostly differences in the way inactive developers are filtered ([2]) or number of reports used (we used 3,000 to compare with the proprietary project, [5] considered 11,311). Looking at the related works in the application of automated bug triaging within industrial context (mainly [7]), there are many characteristics that proprietary projects do not typically share with open source software and that can be at the basis of different results. Within our company-based collaboration, we particularly noticed the needs to discuss: i) the number of issues per developers before having reliable predictions, ii) the number of top-x recommendations to present, iii) the implications for online learning, as well as iv) providing also support for teams assignment in parallel with individual one.

References

[1] G. C. Murphy and D. Cubranic, "Automatic bug triage using text categorization," in *the Sixteenth Int. Conference on Software Engineering & Knowledge Engineering*. KSI Press, 2004, pp. 92–97.

[2] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 361–370.

[3] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 3, pp. 309–346, Jul. 2002.

[4] S. N. Ahsan, J. Ferzund, and F. Wotawa, "Automatic Software Bug Triage System (BTS) Based on Latent Semantic Indexing and Support Vector Machine," *2009 Fourth International Conference on Software Engineering Advances*, pp. 216–221, Sep. 2009.

[5] M. Alenezi, K. Magel, and S. Banitaan, "Efficient Bug Triaging Using Text Mining," *Journal of Software*, vol. 8, no. 9, pp. 2185–2190, Sep. 2013.

[6] X. Xia, D. Lo, X. Wang, and B. Zhou, "Dual analysis for recommending developers to resolve bugs," *Journal of Software: Evolution and Process*, vol. 27, no. 3, pp. 195–220, 2015.

[7] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, "Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts," *Empirical Software Engineering*, pp. 1–46, 2015.

[8] H. Hu, H. Zhang, J. Xuan, and W. Sun, "Effective bug triage based on historical bug-fix information." in *ISSRE*. IEEE Computer Society, 2014, pp. 122–132.

[9] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[10] N. K. Singha Roy and B. Rossi, "Towards an improvement of bug severity classification," in *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*. IEEE, 2014, pp. 269–276.