Identifying Bugs with Incorrect Priorities

Wes Szamotula, Boyuan Feng, Paul Bennett

December 2016

Abstract

Maintaining accurate bug priorities is an important part of software maintenance. Assigning the appropriate priority is not always easy, and these priorities can change as bugs are investigated. To improve this process, we have applied naive Bayes and logistic regression to identify bugs with the incorrect priority. These algorithms were trained on 3,000 bugs from the Mozilla Firefox bug repository submitted over the last two years. Using the bug change history we created snapshots of each bug when they were one week old, and converted the summary and comments into a weighted bag-of-words vector. Approximately 2% of bugs had the incorrect priority since they were re-prioritized at some point in the future. We achieved 5% precision and 50% recall in predicting these incorrectly prioritized bugs.

1 Introduction

Accurately prioritizing bugs is very important in software development. There are often many bugs in code and developers usually do not have enough time to fix all of them. Identifying which ones should be fixed is essential to good maintenance of software. This is usually done through assigning a priority to each bug, and focusing development on the highest priority bugs. However, assigning the appropriate priority is not a trivial task and depends on many factors such as severity, scope, and complexity. If a bug is assigned the incorrect priority it might not get fixed soon enough, or could delay a developer from fixing something more important.

When a bug is initially reported there is often limited information. As developers and testers complete their investigation, they will document more information in the discussion linked to the bug. This information can be used to determine if the initial priority assigned to the bug was actually correct. We have analyzed this information in 24,000 bugs submitted for Mozilla Firefox from 2015 through November 2016. Using the change history of each bug, we are able to create snapshots at a set age and analyze the summary and comments using bag-of-words. With naive Bayes and logistic regression machine learning algorithms, we have developed a predictor that identifies bugs with an incorrect priority. We have achieved 5% precision and 50% recall with these methods.

2 Prior Research

2.1 Bug ranking

Kanwal and Maqbool [1] used SVM to analyze a bug dataset and predict the initial priorities. They achieved a precision and recall of about 46%. Using bug databases from five different projects, Sharma et. al [2] trained several different machine learning algorithms. The accuracy of these algorithms were around 70%. Kim and Ernst [3] tried to prioritize bugs by their lifetimes. They determined the average lifetime for each bug category, and made the assumption that bugs that were fixed quickly were high priority bugs. They ranked the bug categories by their average lifetime and used the rankings to prioritize new bugs based on their categories.

2.2 Fix time for bugs

Giger, Pinzger and Gall [4] tried to determine if it is possible to predict the lifetime of a bug, given the use of existing information in a bug repository. Using a data set from Bugzilla, the authors implemented several machine learning algorithms and compared their prediction precisions. The author discretized the time in days to seven categories to create a classification task. The most useful algorithm was logistic regression, which correctly classified about 35% of bugs. Instead of predicting exactly the lifetime of a bug, Panjer [5] separated bugs into quickly and slowly fixed. To predict

which group new bugs should fall in they used decision trees computed with the Exhaustive CHAID algorithm.

Weiss et. al [6] used a kNN algorithm to estimate the fix time for a bug. As a new issue report is entered into the bug database, they would search for existing issue reports with the most similar descriptions, and combine their reported fix time as a prediction for the fix time of the report. This method achieved an accuracy of around 25%.

2.3 Assigning developers to bugs

Zhang et. al [7] used Latent Dirichlet Allocation (LDA) to extract topics from historical bug reports. This helped identify which topic a new bug report belonged to. This let them assign the new bug to developers who had solved bugs of this type before. By analyzing the relationships between the developers and reporters whose bug reports attracted the most number of comments, they were able to validate a developer's experience fixing bugs.

2.4 Identifying duplicate bugs

Zhou and Zhang [9] and Sun et. al [8] identified textual features of bug reports and used machine learning methods to identify similarities between the new bug and old bugs in the database. Then the algorithms would suggest the top-10 similar bugs. Generally, the recall rate for the top 10 retrieved bugs was between 70% and 80%.

3 Methodology

3.1 Data Source

We used the rest API of Bugzilla to download the 24,000 bugs that were created between January 1, 2015 and November 16, 2016. After downloading these bugs we performed some analysis on the data. We found that most bugs only have a couple comments (fig 1). Furthermore, we found that the majority of bugs were un-prioritized (fig. 2).

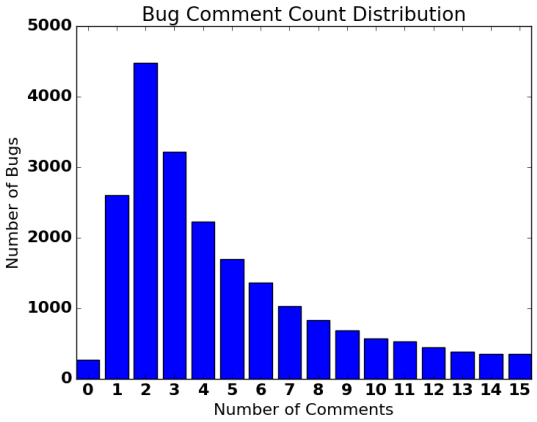


Figure 1: Bug Comment Count Distribution

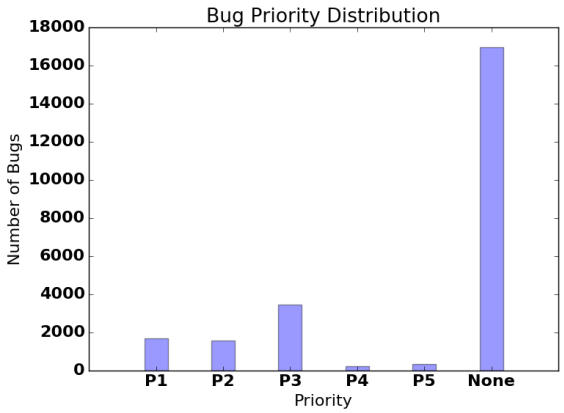


Figure 2: Bug Priority Distribution

In the real world, a bug classifier would be applied to ongoing bugs rather than completed bugs. Therefore we wanted to train and test our classifier on ongoing bug data. To achieve this we converted each bug into a snapshot from when it was 7 days old. To create these snapshots we used the history of changes and reverted each one in reverse chronological order. We also removed any comments that were added more than 7 days after the bug was created.

After creating our snapshots of the data we filtered the bugs based on several criteria. We removed any bugs that had already been resolved, since those priorities would not change in the future. We also removed any bugs that hadn't been assigned an initial priority, since we wanted to focus on bugs where the priority changed from one value to another. Finally, we removed bugs that didn't have any comments, since there had been no additional investigation into these bugs for us to analyze. After this filtering process, we were left

with 3000 snapshots with 2% re-prioritized in the future.

3.2 Prediction Task

Bugs in the Mozilla bug database are assigned priorities P1 through P5. Instead of predicting which of the five priorities a bug would be reassigned to, we instead classified bugs on whether the priority would change or stay the same. We chose to make the task binary for two reasons: 1) it allowed us to apply certain analysis such as area under the receiver operating characteristic that we would not have been able to apply otherwise. 2) The binary classification agrees more with the anticipated application of the algorithms. We anticipate algorithm would be used to flag potential changes, but a reviewer would still assign a priority.

3.3 Feature Set

We used the bag-of-words method to translate the textual data into feature vectors. Thus each feature vector consisted of n elements, where n is the total number of distinct words in the training set. The i^{th} element of the feature vector represents the number of times word i appeared for a given bug snapshot. The initial priority was also used as a feature. This was done by simply appending the numeric priority to the end of the bag-of-words vector. We then used term frequency-inverse document frequency (tf-idf) to weight each entry in the bag-of-words vector. Tf-idf increases proportionally to the number of times a given term appears in a document (bug report), but is offset by the frequency of the term in the corpus (all bug reports). This has the effect of down-weighting common words that are not informative such as 'the' and 'is', while up-weighting words that are more unique to a given bug report.

3.4 Naive Bayes and Logistic Regression

Naive Bayes was used initially as our algorithm. This is a common baseline algorithm which generally has good success in text-based classification tasks.

After obtaining results for the naive Bayes classifier, we also used logistic regression to predict whether a bug would be re-prioritized. Michael Jordan and Andrew Ng's paper [10] discusses the differences between logistic regression and naive Bayes. Naive Bayes has a larger asymptotic error than logistic regression and converges faster. Since we had a large dataset, speed of convergence was not a factor. Therefore using logistic regression was a natural next step.

3.5 Stratified 10-fold Cross Validation

10-fold cross validation was used to validate our results. We separated our data to ten equal-sized subgroups. Each training iteration, we fit our model on nine subgroups and use the left-out one to calculate precision and recall. Finally, we use the average precision and recall over all folds as the final precision and recall. Because the problem is heavily skewed, we used stratified 10-fold cross-validation which equally distributes classes between folds. This prevents having folds with very few or no positive examples

4 Results

4.1 Top-10 Correlated Words

We calculated the correlation between each word and the label, which influences whether a bug will be re-prioritized or not. Table 1 presents the top-10 correlated words. "Intermittent" was the word most often associated with a priority change. This makes sense because a bug that only occurs intermittently is more difficult to prioritize than bugs that are easily repeatable. Another term that is strongly correlated with bugs being re-prioritized is "nigelbabu". This is actually a developer, who contributes heavily to Bugzilla. "rev4" is a mini PC related with Mac OS. "snow" and "leopard" could be combined into "snow leopard", which is a Mac OS.

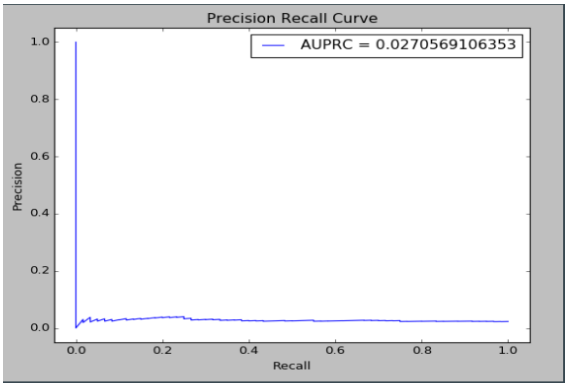
Word	Count	Correlation
intermittent	151	0.355
nigelbabu	17	0.313
rev4	44	0.297
leopard	44	0.297
snow	87	0.294
info	6664	0.294
r4	43	0.291
browser	6605	0.271
xi	52	0.264
test	8701	0.258

Table 1: Top 10 Correlated Words

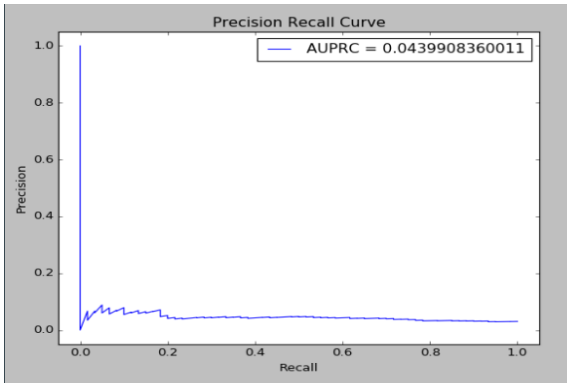
4.2 Naive Bayes and Logistic Regression Classifiers

Predicting bugs with incorrect priorities by identifying bugs that are re-prioritized in the future was a difficult task. We achieved the best results with logistic regression and obtained a precision of 5% and recall of 50%. The precision and recall curves for these two methods can be seen in figures 3a and 3b. The curves are very flat, and we can only gain slight improvements to the precision by taking a lower recall.

The receiver operating characteristic curves for these two classifiers show the difference between their results. This data can be seen in figures 4a and 4b. The logistic regression classifier is generally able to achieve a better true positive rate at varying thresholds.

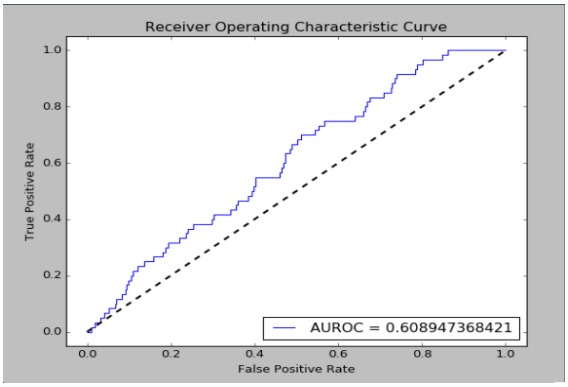


(a) Naive Bayes

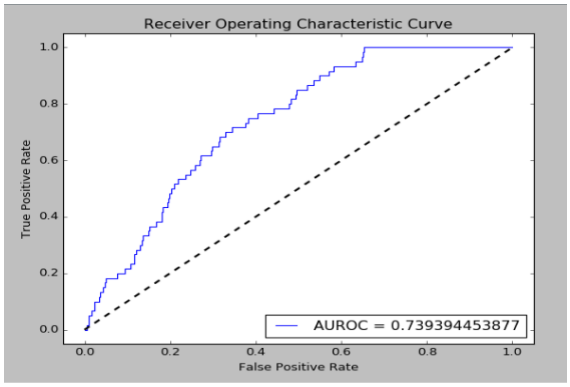


(b) Logistic Regression

Figure 3: Precision Recall Curves with 7 day snapshots and 1 comment threshold



(a) Naive Bayes



(b) Logistic Regression

Figure 4: Receiver Operator Characteristic curves with 7 day snapshots and 1 comment threshold

4.3 Results for varying filter criteria

When designing our algorithm we chose to create 7 days snapshots and require each snapshot to have at least 1 comment. We chose a comment threshold of 1 since most bugs had few comments. Using a higher comment threshold would have made our training dataset too small. We chose 7 days for our snapshot length so the bugs would have had time to be investigated. After a week the developers would be able to gather more information about the impact, complexity, and scope of the problem and make better priority assignments. To validate our settings we tested how our results varied when we changed these numbers.

When we modified the length of our snapshots, we found that shorter snapshots were re-prioritized more often. As shown in figure 5 when our snapshot

length is 1 day about 6% of bugs are re-prioritized at some point in the future. This rate decreases as the snapshot length remains around 2% after one week. During the first week of a bug's lifetime its priority tends to be a little more volatile than after one week.

We found that the area under our ROC curve for logistic regression with a 1 comment threshold tended to increase slightly as we increased the snapshot length as shown in figure 6. This means the algorithm tended to do a better job at making predictions for older bugs. This matched our expectations as bugs at this point should be more thoroughly investigated and the comments should have more useful information.

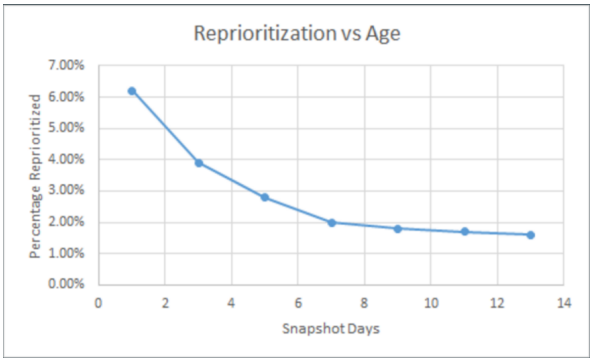


Figure 5: Percentage of filtered snapshots re-prioritized for each snapshot length

We also tried modifying our comment threshold, but this was not as informative. Most bugs only have 2 comments, so once we increased the comment threshold our number of snapshots greatly decreased. This left us with too few instances to learn a reasonable classifier.

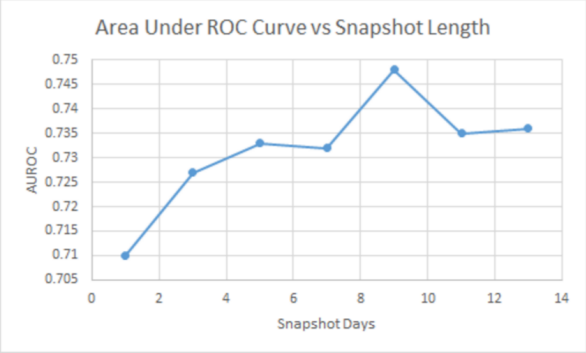


Figure 6: Area under the ROC curve for logistic regression with 1 comment threshold at varying snapshot lengths

5 Future directions

5.1 Improving Classifiers

There are several different directions we would like to take this research in the future in order to improve the precision and recall of our classifiers. Using different machine learning algorithms or ensemble methods may be able to improve our results. Some past work had good success with support vector machines and neural nets at using bug summaries to predict priorities [2], so those methods would be a good place to start. Using different feature representations may also be able to improve our results. There is a lot of other information in bug reports or learn-able through other algorithms like affected code, estimated fix time, products, and sub-products. Using this data in addition to the comments may lead to better predictors.

We approached this as a binary classification problem to predict bugs that are re-prioritized in the future. Past approaches have had decent success estimating initial priority of bugs. A better way to approach this problem may be through using a classifier that predicts specific priorities. If the bug's priority does not match the predicted priority, it could be suggested for re-prioritization. Bugs were re-prioritized at a very low rate in practice, and this method would not depend on bugs actually being re-prioritized, so it would have a less skewed data set.

Exploring more data sets could also have an impact on our results. We just used bugs from one open source application over the last couple of years. Since so few bugs are re-prioritized in practice, expanding the data set by a few years would give us many more re-prioritized examples to study. These additional bugs might be able to improve the accuracy of the method. Additionally, exploring other software could lead to more interesting discoveries. Bug priorities may be set in different ways for other open source software or closed source software.

5.2 Prioritization in the Real World

The way bugs were actually prioritized and re-prioritized at Mozilla Firefox did not match our expectations. We found that the majority of bugs are never assigned a priority, and very few bugs are re-prioritized after the initial priority is assigned. It would be surprising if users only make about 5% errors in assigning the initial priority of bugs. It may be that the accuracy of priorities assigned to bugs is not as important in practice. A study of how

companies set bug priorities and re-prioritize them in the real world could reveal interesting insights about this practice.

6 Conclusion

The way bug priorities are set in practice at Mozilla Firefox did not match our initial expectations. Many bugs never get a priority assigned, and once a priority is assigned it is rarely changed. Only 2% of bugs were re-prioritized in practice and this low prioritization rate made this a difficult and heavily skewed classification task. We used the text from the bug summaries and comments processed through tf-idf weighted bag-of-words vectors for our feature representation. We used naive Bayes and logistic regression to predict if bugs would be re-prioritized in the future since these methods have had good success with natural language tasks. However, we were only able to obtain 5% precision and 50% recall in predicting bug re-prioritization.

While we did not have much success with our methods, these results could possibly be improved. The use of other algorithms, feature representations, or classification tasks could have a large impact on these results. Before exploring other methods, a more general study of how bug priorities are used in practice would be beneficial. The way priorities are used did not match our expectations, and they probably take on a different meaning at different companies. Gaining a better understanding of how they are used could drive future research in this area in a better direction.

References

[1] Kanwal, J., & Maqbool, O. Managing open bug repositories through bug report prioritization using SVMs. *Proc. of the Int. Conf. on Open-Source Systems and Technologies, Lahore, Pakistan.* (2010)

[2] Sharma, M., Bedi, P., Chaturvedi, K. K., & Singh, V. B. Predicting the priority of a reported bug using machine learning techniques and cross project validation. *2012 12th Int. Conf. on Intelligent Systems Design and Applications (ISDA)* 539-545. (2012)

[3] Kim, S., & Ernst, M. D. Prioritizing warning categories by analyzing software history. *Proc. of the Fourth Int. Workshop on Mining Software Repositories* 27. (2007)

[4] Giger, E., Pinzger, M., & Gall, H. Predicting the fix time of bugs. *Proc. of the 2nd Int. Workshop on Recommendation Systems for Software Engineering* 52-56. (2010)

[5] Panjer, L. D. Predicting eclipse bug lifetimes. In *Proc. of the Fourth Int. Workshop on mining software repositories* 29. (2007)

[6] Weiss, C., Premraj, R., Zimmermann, T., & Zeller, A. How long will it take to fix this bug?. *Proc. of the Fourth Int. Workshop on Mining Software Repositories* 1. (2007)

[7] Zhang, T., Yang, G., Lee, B., & Lua, E. K. A novel developer ranking algorithm for automatic bug triage using topic model and developer relations. *2014 21st Asia-Pacific Software Engineering Conf.* 223-230. (2014)

[8] Sun, C., Lo, D., Khoo, S. C., & Jiang, J. Towards more accurate retrieval of duplicate bug reports. *Proc. of the 2011 26th IEEE/ACM Int. Conf. on Automated Software Engineering* 253-262. (2011)

[9] Zhou, J., & Zhang, H. Learning to rank duplicate bug reports. *Proc. of the 21st ACM Int. Conf. on Information and knowledge management* 852-861. (2012)

[10] Jordan M. & Ng A. . On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. *Advances in neural information processing systems* 14, 841. (2002)