ANALYSIS AND PREDICTION OF BUG DUPLICATES IN KDE BUG TRACKING SYSTEM

Gregor Leban
Artificial Intelligence Laboratory
Jozef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
Tel: +386 1 477-53-61
e-mail: gregor.leban@ijs.si

ABSTRACT

Bug tracking systems (BTS) are systems that allow users of some software to report to developers bugs they encountered while using it. Common problem of BTS are duplicated reports of the same bug. Since identifying bug duplicates is a time consuming task we show in this paper an approach to automatically identifying duplicates using text-mining methods. We demonstrate the usability of our method on KDE Bugzilla BTS which contains 249,083 bug reports of which 47,093 are duplicates.

1 INTRODUCTION

Software developing companies and organizations very often use bug/issue tracking systems (BTS) such as Bugzilla[1], Mantis[2] or LaunchPad[3]. Using such systems, users can report to the developers the issues they encountered while using the software. Bug reports consist of a structured and unstructured part. The structured part of the report contains the name of the product where the problem occurred, the component's name, status of the bug, priority and severity. The unstructured part contains the summary of the bug and a description. The provided information should ideally be enough for the developer to identify the problem and fix it. People can also post comments to bug reports where they can clarify the problem or provide additional information or solution to the problem. It is common for a bug to have tens of comments.

One of the difficulties with bug reports is that they are reported by users who don't have an overview of the existing bug reports. As a consequence, many reports describe an issue that has already been reported sometime in the past. Such redundant reports are called bug duplicates. The reason they are problematic is that a lot of time is wasted because of them. They falsely appear to provide information about a new issue and can cause different developers to not-knowingly work on fixing the same bug. To avoid such loss of time a bug is often first inspected if it is a duplicate. If a bug is identified as such, bug tracking systems allow users/developers to explicitly mark them as duplicates to let others know that they can be disregarded. Identifying a bug as a duplicate is however a difficult and imprecise task which requires a lot of time.

In order to alleviate the problem of manually identifying bug duplicates ALERT[4], an EU project that aims to help open source communities to more efficiently manage projects, plans to implement automatic methods that will be able to determine if a given bug is a duplicate. In this paper we will present our first experiments in this area which show promising results.

We will start by introducing the bug tracking system that we used in our experiments and describe the way in which the data was processed. Next we will present the algorithm used to identify the duplicates and show its performance. We will conclude with a summary and some ideas for future work.

2 KDE BUG TRACKING SYSTEM

KDE[5] is an international software community that is developing a set of free, cross-platform applications. They have more than 1,800 developers who have created more than 6 million lines of code. We selected KDE as our case study because they are partners in the ALERT project.

KDE uses Bugzilla BTS to track bugs. KDE started using Bugzilla in 1999 and until August 2010 249,083 bugs were reported. On average there are almost 2,000 bugs reported per month. As for most projects, the number of duplicate reports represent a significant percentage of the repository. In the KDE repository, almost every fifth report is a duplicate. What is even worse is that the ratio of duplicates seems to increase over time and has in the past already reached the value of 0.42.

Out of 249,083 bugs 47,093 were manually marked as bug duplicates. In most cases there is only one duplicate of a bug. Figure 1 shows that there are more than 10,000 such reports. Similarly, there are 3,000 cases where there are two duplicates of one bug. As we can see, the numbers quickly decrease, although we can even find a bug that has 251 duplicated reports.

2.1 Importing the data

In order to be able to analyze the data and predict the duplicates we imported the content of KDE Bugzilla into

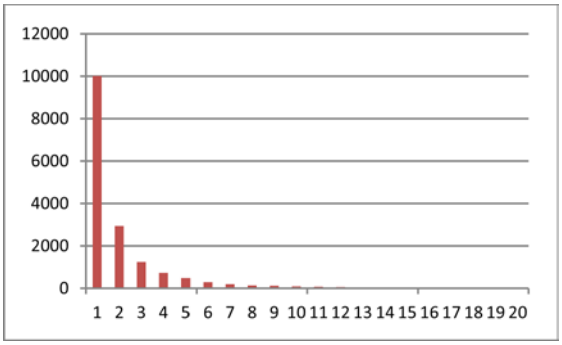


Figure 1: *Distribution of the number of times the same bug was reported. For example, in 3,000 cases there were two duplicate reports created for the same bug.*

Contextify[6]. For each bug report we treated the initial description of the bug and all the following comments as individual documents – in the same way as the data is also stored in Bugzilla. Each of these documents was stored in Contextify in the form of bag of words. As a part of the pre-processing we also ignored the stop words and stemmed the words using the Porter stemmer.

3 PREDICTING THE DUPLICATES

There are different tasks that we can identify regarding the prediction of bug duplicates. One task could be to build a model that would be able to classify a given bug report as a duplicate or non-duplicate. On the surface our problem seems like a typical binary classification problem. Each bug report in the repository represents one learning example, the words in the report are the attributes and the value of the class is 1 or 0, depending on whether the bug has a marked duplicate or not. However, if we think a bit about this definition of the problem we can see that it is not very sensible. Words themselves don't carry any valuable information that would enable us to separate reports into two classes. Reports that mention, for example, the word "kmail" are no more likely to be duplicates than the reports that mention "gnome". If such differences do exist they can only be accidental and using them would only lead to an over-fitted model. The classification model could potentially be improved by adding as attributes the available meta-data that is present in bug reports. By observing the author of the bug report, for example, the model could perhaps identify a group of people who more often than others submit bug duplicates. Since people usually don't intentionally submit bug duplicates such discoveries are unlikely and would not significantly improve the accuracy of the model.

As an alternative option for building a classification model we could also consider the following scenario. Each bug is again a learning example and the words in the report are the attributes. In this case, each bug has a different class value except the duplicates. All the duplicates of a particular bug would have the same class value as the original bug report. The classification model built on such data could predict that a new bug report is the duplicate of an existing bug. If the prediction would be probabilistic we could say that if the probability of the most likely class is below a certain threshold then the bug is not a duplicate. There are two related problems with this approach. As the number of reported bugs increases so does also the number of possible classes. Having 200,000 possible classes is unacceptable since there are no methods that could build a reliable model with so many classes. Also, in order for a method to build an accurate model it has to generalize the learning examples. In this scenario, however, most of the class values only have one learning example (exceptions are bug duplicates) which doesn't allow us any generalization. Classification models built in this way are consequently also bound to be inaccurate.

What is it therefore that we can do with this data? What we can is for a given bug to successfully identify other bug reports that are similar and get a numerical value of this similarity. In this case we can't say that the given bug *B1* is a duplicate but we can say, for example, that the most similar bug to *B1* is *B2* and that similarity between them is 0.56. Such result is not as useful as classification would be, but a list of most similar bugs would still be very helpful for the person who is about to commit a new bug report. Potentially we could also set a threshold for similarity and say that the bug is a duplicate if the similarity exceeds the selected threshold.

3.1 Computing similarities between bug reports

As stated before, each bug report contains an initial description of the bug and potentially any number of comments. Since we expect that the comments can contain valuable additional information about the bug we decided to concatenate the subject of the bug, the initial description and all the comments into one report and to treat this as a single document when we import bugs into Contextify. Documents are in Contextify represented using the vector space model. Each term in the document is weighted using the TF-IDF weighting scheme. We computed term frequency (TF) and inverse document frequency (IDF) for term t_i in document d_j as:

$$TF_{i,j} = f_{i,j} \qquad IDF_i = \log \frac{N}{n_i}$$

where $f_{i,j}$ is the frequency of term t_i in document d_j, N is the number of all documents and n_i is the number of documents that contain t_i. There are several variants of term-weighting and we decided to compute TF-IDF weight $w_{i,j}$ simply as:

$$w_{i,j} = TF_{i,j} \times IDF_i$$

In order to compute similarities between bug reports we also need a measure that would evaluate the correlation between any two reports. We used *cosine similarity* which is the standard measure for quantifying this correlation. For documents d_k and d_l we computed the similarity $sim(d_k, d_l)$ as:

$$d_k = [w_{1,k}, w_{2,k}, \dots, w_{M,k}]$$
$$d_l = [w_{1,l}, w_{2,l}, \dots, w_{M,l}]$$
$$sim(d_k, d_l) = \frac{\overrightarrow{d_k} \cdot \overrightarrow{d_l}}{|\overrightarrow{d_k}| \times |\overrightarrow{d_l}|}$$

3.2 Comparison of similarities between duplicates and non-duplicates

Using the described measure we can now compute similarity between any two bug reports. The question that now arises is how well can this similarity be used to detect bug duplicates – in other words, do bug duplicates really use more similar words in their descriptions than non-duplicates?

To answer this question we performed the following experiment. First we computed for all bugs that don't have duplicates and are not marked as duplicates what is the similarity of their most similar bug report. We then performed a similar computation on bugs that have duplicates (we'll call them original bugs) or are marked as a duplicate. We created sets of bugs where each set consisted of one original bug and all its duplicates. For each bug in the set we computed similarities with other bugs in the set and remembered the maximum value. A graph displaying probability density function of similarities for these two groups of bugs is displayed in Figure 2. As it can be seen, the curves are similar, although the curve for non-duplicates is shifted more to the left and has lower density for higher values of similarity. Based on the graph we can conclude that the similarities are higher between duplicated reports but there is no good threshold that would allow us to accurately classify the bug as duplicate or not based on the highest similarity.

3.3 Ranking bug reports based on similarity

Although classification based on similarity is not accurate we can still help the users to identify bug duplicates. For a selected bug report we can compute a ranked list of most similar bug reports. The user can then inspect the list and decide if the bug is a duplicate or not. If the ranking is good it would be enough for the user to check only the first few reports in order to decide if the report is a duplicate or not.

To test how much can the ranking help the users to identify the duplicates we performed an experiment. For each bug report that has duplicated reports we computed a ranked list of 100 most similar bug reports in the whole repository. We then checked how well are the duplicated reports of the bug ranked in this list. This is a standard information retrieval task where the query is the tested bug report and the answer set is the set of similar reports.

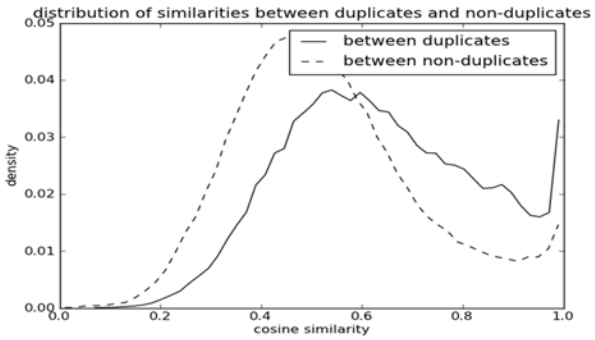


Figure 2: *The comparison of similarities between duplicated reports and reports without duplicates.*

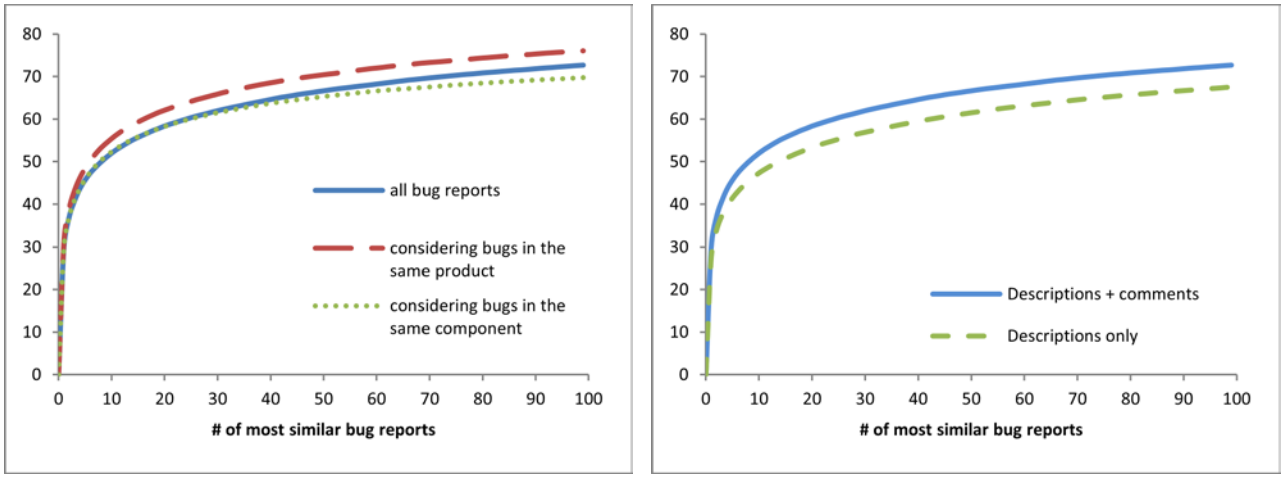
There are different metrics that can be used to evaluate the success of this task. Commonly used measures are precision and recall. They are not the most appropriate for us since we (1) are only interested where in the list is the *first* correct answer (duplicate) and (2) there is most often only one correct answer (most bugs have only one duplicate) which would automatically result in low precision. Instead we decided to use mean reciprocal rank that is often used in question answering systems. Reciprocal rank is the inverse of the rank of the first correct answer and mean reciprocal rank (MRR) is the average of the reciprocal ranks for a sample of queries *Q*:

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$

Our set of queries *Q* consisted of 63,861 reports (duplicated + original bug reports) and the computed MRR was 0,374. The Figure 3.a shows the percent of detected duplicates in relation to the number of inspected reports. The full line shows the results we obtained by comparing the test reports with all other bug reports in the repository. We can see that the curve is very steep and by looking at the first 5 most similar bug reports we can detect more than 45% of all duplicates.

In order to additionally improve the ranking we wanted to see if we can use some information from the meta-data of the reports. For each bug report, the user has to specify to which product it belongs and then even more specifically to which component inside the selected product. The information about the selected product and component is then stored as meta-data of the report. Our expectation is that we can use this information to improve the ranking by only considering those reports that are assigned to the same product/component. The influence of this information on ranking is also shown in Figure 3.a. Interestingly we can see that using the product information improves the ranking, while using the component information has a detrimental effect. The reason for this is that users sometimes assign the bug to the wrong product/component. In Bugzilla, there are almost 5,000 duplicates that are assigned to the wrong product and more than 12,000 that are assigned to the wrong component. If we only consider reports within the same product/component we therefore cannot locate the duplicate for these reports which in turn degrades the ranking quality.

We were also interested in knowing if the initial bug descriptions contain all the necessary information needed to identify the duplicates or do the following comments also contribute something valuable. For this we again for all duplicated reports computed 100 most similar reports in the whole repository, but this time the documents representing the reports consisted only of the bug subjects and the initial



(a) (b)

Figure 3. *The percent of detected duplicates in relation to the number of inspected reports. The figures show how the use of the meta-data (a) and the use of comments in the reports (b) influences the accuracy of the ranking.*

descriptions (without comments). Results are shown in Figure 3.b. It is evident that the ranking without using comments is worse which indicates that comments are valuable and should also be used when identifying duplicates.

4. Related work

The presented work does not present any new methods but only demonstrates the applicability of existing approaches on a special use case. The most related work was done by Hiew[7] who searched for similar bugs by first computing centroids of related bugs. Čubranić and Murphy's Hipikat[8] project determines which reports in a repository are similar to each other using an information retrieval algorithm. Wang describes another approach to detecting bug duplicates using natural language and execution information[9]. Automatic detection of duplicate documents has also been considered in other contexts. In large document collections, for example, the duplicates are identified to maintain the speed of search engines[10].

5 CONCLUSION AND FUTURE WORK

We have presented our preliminary work in the field of automatically identifying bug duplicates. First we have described the details of the KDE BTS that was used as our case study. Next, we described the preprocessing steps that we used to represent the bug reports in the vector space model. We defined the similarity measure used and evaluated how accurately we can rank duplicated bug reports.

We have several ideas for future work. First we plan to inspect the reports that have high similarity and are not marked as duplicates. We will identify what is the common property of these reports (for example, they might be very short reports, or might contain common phrases that are computer generated, like "no debugging symbols found") and try to take this information into account when weighting terms or computing similarity between reports. Another idea is to check how informative the date of the reported bug is. Since new bugs occur with new releases of the software it is probably more likely that two reports are duplicates if they are closer in time. We will also try to improve the use of the meta-data like product/component information. Instead of considering only reports in the same product we can soften the constraint to include also related products.

6 ACKNOWLEDGMENTS

This work was supported by the Slovenian Research Agency, European Social Fund and ALERT (ICT-2009.1.2).

References

[1] "Bugzilla," 2011. Available: http://www.bugzilla.org/.
[2] "Mantis," 2011. Available: http://www.mantisbt.org/.
[3] "LaunchPad," 2011. Available: https://launchpad.net/.
[4] "ALERT," 2011. Available: http://www.alert-project.eu/.
[5] "KDE," 2011. Available: http://www.kde.org/.
[6] G. Leban and M. Grobelnik, "Displaying email-related contextual information using Contextify," International Semantic Web Conference, Shanghai, China, 2010, pp. 181-184.
[7] L. Hiew, "Assisted detection of duplicate bug reports," The University Of British Columbia, 2006.
[8] D. Cubranic and G. C. Murphy, "Hipikat: Recommending pertinent software development artifacts," 2003.
[9] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 461–470.
[10] A. Chowdhury, O. Frieder, D. Grossman, and M. C. McCabe, "Collection statistics for fast duplicate document detection," *ACM Transactions on Information Systems (TOIS)*, vol. 20, no. 2, pp. 171–191, 2002.