Exploiting Hubs for Self-Adaptive Secondary Re-Ranking In Bug Report Duplicate Detection

Nenad Tomašev, Gregor Leban, Dunja Mladenić
Artificial Intelligence Laboratory
Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia
e-mail: nenad.tomasev@ijs.si, gregor.leban@ijs.si, dunja.mladenic@ijs.si

Abstract. *Bug duplicate detection is an integral part of many bug tracking systems. Most bugs are reported multiple times and detecting the duplicates saves time and valuable resources. We propose a novel approach to potential duplicate report query ranking. Our secondary re-ranking procedure is self-adaptive, as it learns from previous report occurrences. It is based on the analysis of temporal evolution of the underlying distribution of influence. The experiments show definite improvements in system performance.*

Keywords. Bug duplicates, issue tracking, ranking, hubs.

1. Introduction

Issue tracking systems are useful tools for software maintenance that allow the users to either report bugs or suggest new features that would extend the existing system functionality.

There are many existing bug tracking systems. Some of the best known include Bugzilla, JIRA, Launchpad and Redmine[1]. There are several types of information that a user is usually able to report to such systems. Apart from the textual description, a user can leave his/hers contact information, specify the type of the request (eg. bug, suggestion), specify the component where the issue was noticed, the build version, the platform on which the system was running, the priority with which the issue needs to be resolved, etc.

If the issue can be replicated, it is verified by the responsible person and needs to be assigned to the engineer who would be working on the fix. Before the assignment takes place, the system needs to check if the same issue had previously already been reported, as there is no need to forward the issue reports have either already been resolved or are being resolved.

Bug duplicate detection is not an easy task, as different users may formulate their free-text reports in entirely different ways.

1.1 Project Goal and Contributions

Our goal was to propose an approach that helps engineers to detect potential bug duplicates by ranking the existing bug reports according to some similarity score. In this paper we present a secondary re-ranking procedure which fine-tunes the output of the primary ranking system. The system ranking was made to be self-adaptive, as it learns from its own mistakes.

2. Related Work

2.1 Bug Duplicate Detection

Automatic bug duplicate detection is usually based on some sort of similarity-based reasoning. The current report is compared to the previously observed reports and the most similar reports are marked as potential bug duplicates. There are many ways to extract features from the reports and different approaches for ranking, grouping and search are possible.

One of the first information retrieval approaches to duplicate detection was proposed in [4]. Textual clustering was another approach [5]. A natural language

[1] http://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems

processing based duplicate detection method based on both the textual and the execution similarity was used in [6]. Using n-grams is also possible and somewhat more robust to noise and domain-specific term usage [7]. Classifiers can be trained to classify the reports into duplicates, but they currently do not achieve high precision [8]. It is possible to exploit features from user profiles, comments and feedback in order to improve the duplicate detection performance [9]. Modeling the influence of typographical errors and alternate spellings is also beneficial [10]. Recent studies have suggested that the traditional vector space models might be superior to topic-based representational models in this context [11].

2.2 Hubs in High-dimensional Data

Many real world networks are scale-free as the node degree distribution follows a power law [12]. This includes the Internet, protein interaction networks, some social networks, etc. The influence within these networks is mostly contained in a small group of very influential high-degree nodes called *hubs*. Most remaining nodes are low-degree and are called *orphans* or *anti-hubs* [12].

High-dimensional data is known to be quite difficult to handle, due to all the sparsity and predominating empty space. Distances concentrate [13] and querying for relevant similar examples becomes much more difficult. Hence the phrase '*curse of dimensionality'*.

Hubness is an important aspect of the dimensionality curse which implicitly affects many similarity-based methods. It was first observed in the domain of music retrieval and recommendation [14].

Hubs emerge as centers of influence in the *k*-nearest neighbor topology of the data, under many widely used similarity measures [15]. In this context, hubs are very frequent nearest neighbors, they are retrieved in result sets more often than would otherwise have been expected. This may cause some semantic inconsistencies and is usually a detrimental phenomenon.

Textual data is unstructured and of high intrinsic dimensionality. It is known to exhibit high hubness [16][17].

Several hubness-aware data mining and machine learning methods have recently been proposed for classification [18][19], clustering [20], metric learning [21] and data reduction [22].

3. Methodology

In our experiments we used the November 2010 image of the KDE bug repository (https://bugs.kde.org/), which contained 249.129 bug reports. A large portion of these reports were marked as duplicates, 47.061. The reports were filed between 21.1.1999 and 2.11.2010. Bugs in KDE repository are related to 479 different products.

The similarity between the reports was obtained by calculating the similarity between the textual bug descriptions. All the available information was used – the subject line, main description and the associated comments.

The textual data was processed in a standard way, including stemming and stop word removal. Porter stemmer [1] was used for word inflection removal.

We have used the standard vector space model [2] with TF-IDF weighting [3]. Assuming $V = [v_1, v_2 .. v_W]$ is the vocabulary used to form the reports, each report is represented as a $|V|$-dimensional vector of weights. Each weight corresponds to a given word in the vocabulary and is derived from its frequency and the inverse document frequency. This is a well-known textual representation.

Let R comprise the set of all filed issues and denote by $r_i = [r_i^1, r_i^2 \ldots r_i^W]$ and $r_j = [r_j^1, r_j^2 \ldots r_j^W]$ the vectors representing two such bug reports. The similarity between the two reports is then defined as the cosine similarity between the corresponding representational vectors[3].

$$sim_{r_i,r_j} = \frac{\vec{r_i} \cdot \vec{r_j}}{|\vec{r_i}| \cdot |\vec{r_j}|} =$$

$$= \frac{\sum_{p=1}^{W} r_i^p \cdot r_j^p}{\sqrt{(\sum_{p=1}^{W} (r_i^p)^2) \cdot (\sum_{k=1}^{W} (r_j^p)^2)}}$$

Issue reports arrive at some fixed order of precedence. Upon arrival, each report is compared to the previously archived documents to check for possible duplicates. Therefore, if a bug arrives at time **T**, it is only compared to the reports that have arrived at some earlier time **t < T**. All subsequent experiments were performed by taking this into account.

We will use the same ordering for bug report feature representations and their time stamps, so that t_i corresponds to the time of receipt of report r_i. Let $\mathbf{T_f} = \mathbf{t_N}$ be the time of receipt for the last received report.

When the newly received report is analyzed, its cosine similarity towards each earlier report is calculated and the list is sorted in a descending order, so that the most similar earlier report is shown first to the system user. This allows the responsible engineers and easy overview of the possible duplicate reports. By inspecting the list, they will mark some of the suggestions as duplicates.

This process is far from perfect, as it involves manual inspection of many free-text issue reports. Sometimes, duplicates do not get noticed. However, the precision of the automatic duplicate detection software is not high enough to be able to replace actual human participation in the process. This is why bug report duplicate detection remains a semi-automatic process.

4. The Influence of Hub-Reports

Let us begin by introducing some notation. Denote by $N_k^T(r)$ the number of times that the report *r* occurred in the top-*k* similarity queries of the duplicate detection system up until time **T**. We will refer to $N_k^T(r)$ as the **temporal occurrence frequency** of *r*. It is a function that is monotonously non-decreasing with respect to **T**.

Hubness manifests itself as high skewness (standardized 3rd moment) of the occurrence frequency distribution. It is defined as follows:

$$SN_k^T(r) = \frac{{}^1\!/_N \sum_{i=1}^{N} (N_k^T(r_i) - k)^3}{({}^1\!/_N \sum_{i=1}^{N} (N_k^T(r_i) - k)^2)^{3/2}}$$

Positive (right) skewness means that the distribution tail is longer on the right side and that most values are lower than the mean. Figure 1 shows the skewness of the bug report occurrence frequency distribution of our data for different values of *k*. The bug reports exhibit very high hubness.

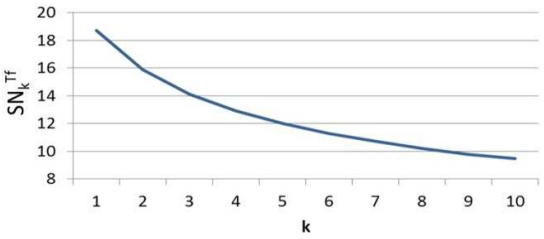

Figure 1: Skewness of the bug report occurrence frequency distribution, indicating high hubness (everything above SN_k of 1-2 is quite high).

The consequences of high occurrence frequency skewness can be seen in Figure 2, where the degree of the major hub is shown, i.e. the maximal $N_k^{Tf}(r)$ over all $r \in R$. We can see that the major hub report occurs very frequently, much more often than it actually occurs as a genuine duplicate. For k=10, the major hub report occurs 1421 times, out of which it is only an actual duplicate once and not a duplicate 1420 times. Out of those 1420 occurrences, in 373 cases an actual duplicate of the query report existed, but it was not the major hub.

As the system can not be expected to always pinpoint the exact duplicate as the most similar document, we are forced to use $k > 1$ in practice. The above given analysis of the nature of hub occurrences suggests

that this might increase the number of false positives.

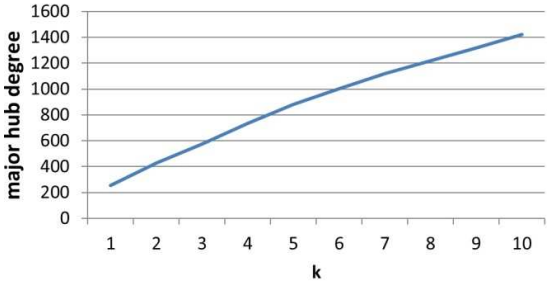

Figure 2: The maximum report occurrence frequency, shown for different values of *k*.

In the context of bug duplicate detection, we define a **bad *k*-occurence** as *each occurrence of report r_i in the query top-k result set of report q such that r_i is not a duplicate of q, if the report q has duplicates in the previously gathered data.* Denote by $BN_k^T(r)$ the temporal **bad occurrence frequency.** The occurrences that are not bad will be referred to as good occurrences and we will denote them by $GN_k^T(r)$,so that $N_k^T(r) = BN_k^T(r) + GN_k^T(r)$.

If a report has no previous duplicates, whatever gets retrieved by the system does not affect its performance. Therefore, we are only interested in those queries where actual duplicates exist.

Most occurrences in the systems turn out to be bad occurrences, as shown in Figure 3. This is not surprising, as the problem of bug duplicate detection is a hard one and systems usually achieve low precision.

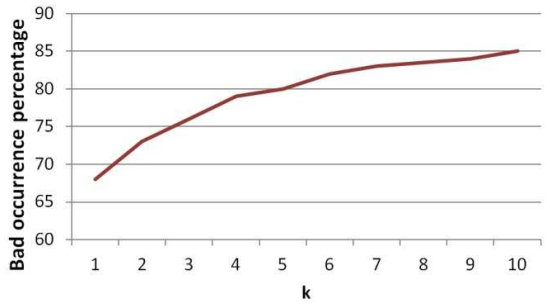

Figure 3: The percentage of bad occurrences in the result sets of the initial bug duplicate detection system.

Our idea was to exploit the information about the previous bad occurrences for each report at any point in time and to favor those reports that have had fewer bad occurrences when creating the final ranking. The secondary re-ranking was based on the temporal *k*-occurrence model.

5. Secondary Re-Ranking and Evaluation

Denote by q_T the currently received report at time **T** for which the query is made to the bug duplicate detection system. Let $R_{q,T} = \{r_{i1}, r_{i2}, \ldots r_{ik}\}$ be the top-*k* result set returned by the system. The system also provides the primary similarity scores.

The secondary hubness-aware re-ranking is performed by re-defining the similarity measure based on the temporal occurrence model. The secondary similarity will be denoted by $sim_k^{H,T}$. After the secondary similarity is calculated, the final ranking is obtained by re-ordering the initial result set based on the new similarity value.

In the temporal occurrence model, the nature of the report occurrence profile might change over time. A good hub might become a bad hub and a false positive as time goes by and the distribution of the data changes, or vice-versa.

Therefore, the total aggregate occurrence count is not a reliable quantity. This is why we have decided to observe only the occurrence counts within a time-dependent sliding window. Denote by $GN_{k,\mu}^T(r)$ the **good truncated occurrence frequency,** that is obtained by counting only among the past μ occurrences of r (or less, if r has occurred less than μ times total). The truncated total occurrence count is then $N_{k,\mu}^T(r) = \min(\mu, (N_k^T(r)+1))$, where we have increased the original occurrence count by one to avoid zero divisions. In a sense, each query report is implicitly included in its own result set at position zero.

The secondary $sim_k^{H,T}$ similarity measure is defined as follows:

$$sim_k^{H,T}(q_T, r) = \frac{GN_{k,\mu}^T}{N_{k,\mu}^T} \cdot sim(q_T, r)$$

This simple formula increases the similarity between the query report and those retrieved reports that have recently had very few bad occurrences. This increases the confidence that the retrieved report is actually relevant for the query and is more likely to be an actual duplicate, if such exists.

Note that it would not be advisable to sort the query set according to the good occurrence percentages alone, as the primary similarity measure is needed to convey the content-wise relevance information.

We have evaluated the effectiveness of the proposed approach by considering the change in two quality indices: the average rank of the first detected duplicate ($rank_{AVG}^{FD}$) and the overall average rank of detected duplicates ($rank_{AVG}^{D}$). A lower rank would indicate and improvement in performance, as it is important to sort the reports in a way that requires least effort from the user. This is why ranking is important. Additionally, re-ranking can also improve the detection rates, if the system operates with two different *k*-values, one for $sim_k^{H,T}$ calculation and the other for top-*k* result retrieval.

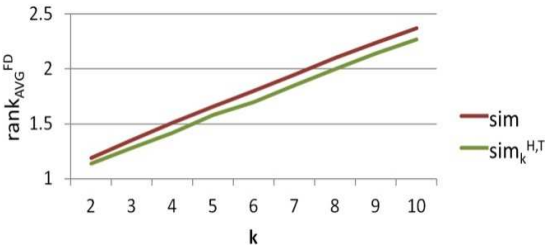

Figure 4: Average rank of the first detected duplicate before and after re-ranking.

The difference in the average first and overall KDE duplicate ranks before and after re-ranking by $sim_k^{H,T}$ is shown in Figure 4 and Figure 5, for different values of *k*. The rank reduction for the first (most similar) duplicate is present for all *k*-values. The improvement in reducing the overall detected duplicate rank increases with *k*.

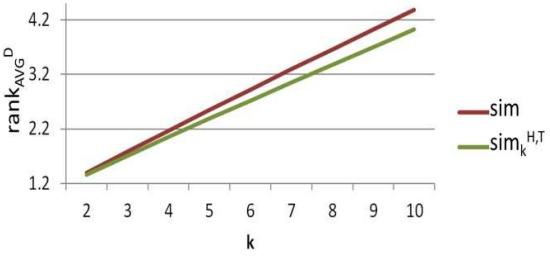

Figure 5: Average rank of the detected duplicates before and after re-ranking.

The proposed ranking system is self-adaptive as it learns the secondary similarity measure by observing and estimating the deficiencies of the primary similarity at each time step, thereby making duplicate reports more similar to each other.

6. Conclusions

Bug duplicate detection is a hard and important problem in software engineering. Automated duplicate detection saves time and money by allowing the engineers to examine bug reports more efficiently.

We have examined the hub-structure of the bug report data and concluded that there are some very frequently occurring reports in the result sets returned by the primary detection system. These reports are being frequently retrieved even when they are not in fact duplicates of the queries.

We have proposed a secondary re-ranking by a secondary similarity measure, in order to reduce the impact of bad hub-reports on the overall performance of the system. The evaluation shows that the proposed solution works well and makes the duplicate reports more similar to each other.

Acknowledgements

This work was supported by the IST Programme of the European Community under ALERT project (ICT-STREP-249119).

References

[1] Porter MF et al. An algorithm for suffix stripping. Program, 1980.
[2] Raghavan VV, Wong S. A critical analysis of vector space model for information retrieval, Journal of the American Society for Information Science 1986; 37(5): 279–287.
[3] Feldman R, Sanger J, editors. The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data. Cambridge: Cambridge University Press; 2006.
[4] Runeson P, Alexandersson M, Nyholm O. Detection of duplicate defect reports using natural language processing. In: Proceedings of the 29th international conference on Software Engineering; 2007. p. 499–510.
[5] Hiew L. Assisted detection of duplicate bug reports. The University Of British Columbia; 2006.
[6] Wang X, Zhang L, Xie T, Anvik J, Sun J. An approach to detecting duplicate bug reports using natural language and execution information. In: Proceedings of the 30th international conference on Software engineering; 2008. p. 461–470.
[7] Sureka A, Jalote P. Detecting Duplicate Bug Report Using Character N-Gram-Based Features. In: Software Engineering Conference (APSEC)*;* 2010. p. 366–374.
[8] Jalbert N, Weimer W. Automated duplicate detection for bug tracking systems. In: Dependable Systems and Networks With FTCS and DCC*;* 2008. p. 52–61.
[9] Liang F et al. Practical Duplicate Bug Reports Detection in a Large Web-Based Development Community. In: Web Technologies and Applications; 2013. p. 709-729.
[10] Banerjee S et al. Handling Language Variations in Open Source Bug Reporting Systems. In: Software Reliability Engineering Workshops (ISSREW); 2012. p. 325-330.
[11] Kaushik N, Tahvildari L. A Comparative Study of the Performance of IR Models on Duplicate Bug Detection. In: Software Maintenance and Reengineering (CSMR); 2012. p. 159-168.
[12] Easly D, Kleinberg J, editors. Networks, Crowds and Markets. Cambridge: Cambridge University Press; 2010.
[13] Aggarwal CC, Hinneburg A, Keim DA. On the surprising behavior of distance metrics in high-dimensional spaces. In: Proceedings of the 8th International Conference on Database Theory (ICDT); 2001. 420-434.
[14] Aucouturier J, Pachet F. Improving timbre similarity: how high is the sky?. Journal of Negative Results in Speech and Audio Sciences 2004.
[15] Radovanović M, Nanopulous A and Ivanović M. Hubs in space: popular nearest neighbors in high-dimensional data. Journal of Machine Learning Research 2011; 1: 2487-2531.
[16] Radovanović M, Nanopulous A and Ivanović M. On the existence of obstinate results in vector space models. In: Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval; 2010. p.186-193.
[17] Tomašev N, Rupnik J, Mladenić D. The role of hubs in cross-lingual supervised document retrieval. In: Proceedings of the Pacific Asian Knowledge Discovery and Data Mining Conference (PAKDD); 2013. p. 185-197.
[18] Tomašev N, Radovanović M, Mladenić D, M. Ivanović. Hubness-based fuzzy measures for high-dimensional k-nearest neighbor classification. In: Proceedings of the MLDM Conference; 2011. 16-30.
[19] Tomašev N, Mladenić D. Nearest neighbor voting in high-dimensional data: learning from past occurrences. Computer Science and Information Systems 2012. 9(2) 691-713.
[20] Tomašev N, Radovanović M, Mladenić D, Ivanović M. The role of hubness in clustering high-dimensional data. In: Proceedings of the Pacific Asian Knowledge Discovery and Data Mining Conference (PAKDD); 2011.
[21] Tomašev N, Mladenić D. Hubness-aware shared neighbor distances for high-dimensional k-nearest neighbor classification. Knowledge and Information Systems 2013.
[22] Buza K, Nanopulous D, Schmidt-Thieme L. Insight: efficient and effective instance selection for time series classification. In: Proceedings of the Pacific Asian Knowledge Discovery and Data Mining Conference (PAKDD); 2011. 149-160.