New Features for Duplicate Bug Detection

Nathan Klein
Department of Computer Science
Oberlin College
Oberlin, Ohio, USA
nklein@oberlin.edu

Christopher S. Corley, Nicholas A. Kraft
Department of Computer Science
The University of Alabama
Tuscaloosa, Alabama, USA
cscorley@ua.edu, nkraft@cs.ua.edu

ABSTRACT

Issue tracking software of large software projects receive a large volume of issue reports each day. Each of these issues is typically triaged by hand, a time consuming and error prone task. Additionally, issue reporters lack the necessary understanding to know whether their issue has previously been reported. This leads to issue trackers containing a lot of duplicate reports, adding complexity to the triaging task.

Duplicate bug report detection is designed to aid developers by automatically grouping bug reports concerning identical issues. Previous work by Alipour et al. has shown that the textual, categorical, and contextual information of an issue report are effective measures in duplicate bug report detection. In our work, we extend previous work by introducing a range of metrics based on the topic distribution of the issue reports, relying only on data taken directly from bug reports. In particular, we introduce a novel metric that measures the first shared topic between two topic-document distributions. This paper details the evaluation of this group of pair-based metrics with a range of machine learning classifiers, using the same issues used by Alipour et al. We demonstrate that the proposed metrics show a significant improvement over previous work, and conclude that the simple metrics we propose should be considered in future studies on bug report deduplication, as well as for more general natural language processing applications.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

General Terms

Management, Reliability

Keywords

Duplicate bug reports, topic model, machine learning

MSR'14, May 31 – June 1, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2863-0/14/05...$15.00
http://dx.doi.org/10.1145/2597073.2597090

1. INTRODUCTION

Software projects with millions of users, such as the Android Operating System, receive hundreds of bug reports every day. Often, multiple bug reports concern the same issue, since users frequently submit bug reports without checking to see if their issue has been mentioned before (or fail to find the relevant reports). Therefore, the automatic grouping of bug reports which concern the same issue is a very helpful tool, and is a topic that has recently received a good deal of attention from researchers. In this paper we are interested in accurately determining whether a pair of bug reports concern the same issue or not, as this is generally the method by which bug reports are deduplicated.

The sophistication of bug report duplicate detection has increased rapidly over the past several years. Jalbert and Weimer [6] used a term-frequency weighting scheme to calculate description and title cosine similarity between bug reports, and was able to automatically identify 8% of duplicate reports from the Mozilla database. Nguyen et al. [9] improved upon this model by combining LDA-based textual comparison, information retrieval, and some categorical features. A recent study by Alipour et al. [2] added contextual data to aid in the creation of topics used by LDA and showed an improvement over a method by Sun et al. [11], using the Android repository as a case study. This study follows the experimental set up of Alipour et al., where bug reports in the repository are randomly paired together and similarity metrics are computed to form data points. The data is then given to a variety of machine learning classifiers to create predictive models. Using a new suite of metrics, our approach was able to predict the correct class of a data point (duplicate or non-duplicate) over 95% of the time using ten-fold cross validation. The metrics proposed in this study have an increase of an average of 8% in accuracy when compared to Alipour et al.'s metrics, and do not require the contextual word lists that Alipour et al.'s metrics do. This allows our study to more easily be applied to a variety of repositories. They show an increase of an average of 15% in accuracy compared to Sun et al.'s metrics. Therefore, we believe our suite of metrics can be used to improve state of the art bug report deduplication efforts, and may have applications to other areas of NLP research as well.

2. METHODOLOGY

Whereas Sun et al. [11] formulated the duplicate bug detection problem as *given a bug report, return a list of the top-k most similar bug reports*, like Alipour et al. [2], we formulate the problem as *given two bug reports, predict whether*

they are duplicates. The latter problem formulation was first used by Lo et al. [7]. As noted by a reviewer, the former problem formulation is more challenging than the latter, because it is easier to differentiate between two random bug reports and two duplicate bug reports than it is to differentiate between two similar bug reports that are not duplicates and two similar bug reports that are duplicates.

We used the data collected by Alipour et al., consisting of reports pulled from the android database between November 2007 and September 2012. In the data set, 1,452 bug reports were marked as duplicates out of 37,627 total. Every bug report has the following features: *Bug ID, Date Opened, Status, Merge ID, Summary, Description, Component, Type, Priority,* and *Version*. Because few reports specify a version, we ignored this final feature.

After collecting the data, groups of duplicate bug reports were placed into buckets. 2,102 unique reports appeared in these buckets. Then, we calculated the topic-document distribution of each summary, description, and combined summary and description for each report using the implementation of latent Dirichlet allocation (LDA) [4] in MALLET [8]. LDA is a probabilistic generative model that creates a specified number of topics based on terms appearing in the corpus, and then generates a topic distribution for each document. LDA was introduced by Blei et al. [4] and has subsequently been used in a variety of text retrieval applications. We used MALLET's default LDA configurations: an alpha value of 50.0 and a beta value of 0.01. We used a 100-topic model, a number suggested by Sun et al.'s results.

Following Alipour et al., we generated 20,000 pairs of bug reports consisting of 20% duplicate pairs. To protect the validity of our study, we ensured that no two pairs contained identical reports.

For each pair, 13 attributes were computed. To stem words for the *simSumControl* and *simDesControl* attributes, we used the Porter stemmer [10]. We used the SEO suite stopword list [1]. LDA distributions are sorted based on the percentage each topic describes, in decreasing order. Table 1 lists the attributes used in the study, and Table 2 shows a pair of duplicate bug reports from the Android data set.

Once all the pairs were created, we tested the predictive power of a range of machine learning classifiers using the Weka tool [5]. Tests were conducted using ten-fold cross-validation. Following Alipour et al., we test the efficacy of a machine learner using its accuracy, the AUC, or area under the Receiver Operating Characteristic (ROC) curve, and its Kappa statistic. The ROC curve plots the true positive rate of a binary classifier against its false positive rate as the threshold of discrimination changes, and therefore the AUC is the probability that the classifier will rank a positive instance higher than a negative instance. The Kappa statistic is a measure of how closely the learned model fits the data given. In this model, it signifies how closely the learned model corresponds to the triagers which classified the bug reports. We used the same machine learners as Alipour et al., and added one additional learner, REPTree with Bagging, or the **B**ootstrap **Agg**regat**ing** technique. Bagging uses iterative training on multiple weighted training sets to prevent overfitting.

The other algorithms used are ZeroR, Naive Bayes, Logistic Regression, C4.5, and K-NN. Note that the ZeroR algorithm simply predicts the mode, and thus is a baseline classifier. For the K-NN classifier, k was set to 7. For the C4.5 algorithm, a confidence factor of 0.025 was used. All other models were set to their defaults in Weka. To compare our results with Alipour et al.'s, we used their "Contextual, categorical, and Labeled-LDA context's data" results, which have the highest average AUC over the five different contextual word lists they tested. To evaluate the performance of the Bagging: REPTree classifier we compared it to the best-performing classifier in the group. Alipour et al. also evaluated the metrics of Sun et al. [11] on this data set, and therefore we will compare our results to Sun et al.'s metrics as well. Finally, we evaluated the information gain of each of our metrics to discover which might account for a change in performance from Sun et al. and Alipour et al. The infor-

Table 1: Attributes for Pairs of Bug Reports

lenWordDiffSum *lenWordDiffDes*	Difference in the number of words in the summaries or descriptions
simSumControl *simDesControl*	Number of shared words in the summaries or descriptions after stemming and stop-word removal, controlled by their lengths
sameTopicSum *sameTopicDes* *sameTopicTot*	First shared identical topic between the sorted distribution given by LDA to each summary, description, or combined summary and description
topicSimSum *topicSimDes* *topicSimTot*	Hellinger distance between the topic distributions given by LDA to each summary, description, or combined summary and description
priorityDiff	{same-priority, not-same}
timeDiff	Difference in minutes between the times the bugs were submitted
sameComponent	Four-category attribute: {both-null, one-null, no-null-same, no-null-not-same}
sameType	{same-type, not-same}
class	{dup, not-dup}

Table 2: Example Bug Reports

Attribute	Bug 21196	Bug 20161
Submitted	oct 25 2011 08:22:51	sep 19 2011 13:05:15
Status	Duplicate	Duplicate
MergeID	7402	7402
Summary	support urdu in android	urdu language support
Description	i just see many description where people continuously requesting google for support urdu in andriod ...	hello i'm unable to read any type of urdu language text messages. please add urdu language in future updates of android ...
Component	Null	Null
Type	Defect	Defect
Priority	Medium	Medium

Table 3: Classification Results

Algorithm	Accuracy %	AUC	Kappa
ZeroR	80.00%	0.500	0.00
Naive Bayes	92.990%	0.958	0.778
Logistic Regression	94.585%	0.972	0.824
C4.5	94.780%	0.941	0.832
K-NN	94.785	0.955	0.830
Bagging: REPTree	**95.170%**	**0.977**	**0.845**

Table 4: Improvement over Alipour et al. [2]

Algorithm	Accuracy	AUC	Kappa
ZeroR	0%	0%	0%
Naive Bayes	+16.741%	+21.57%	+121.78%
Logistic Regression	+7.33%	+7.52%	+38.09%
C4.5	+2.90%	+5.97%	+10.15%
K-NN	+3.59%	+4.83%	+9.77%
Bagging: REPTree	+3.33%	+7.24%	+11.76%

Table 5: Improvement over Sun et al. [11]

Algorithm	Accuracy	AUC	Kappa
ZeroR	0%	0%	0%
Naive Bayes	+18.27%	+23.14%	Undef.
Logistic Regression	+14.19%	+19.41%	+152.76%
C4.5	+12.13%	+31.42%	+92.41%
K-NN	+15.06%	+29.58%	+79.81%
Bagging: REPTree	+12.59%	+25.58%	+83.06%

mation gain of a metric corresponds to the amount by which it reduces the entropy of a set, such that when a decision tree separates data based on a metric with high information gain, the resulting sets have a much lower entropy than the initial set, while separation on a metric with low information gain yields sets with similar entropy. Therefore it is roughly equivalent with how useful a metric is for splitting data in decision tree learners like C4.5.

3. CASE STUDY

We applied these methods to the Android data set obtained by Alipour et al. Table 3 details the accuracy, AUC, and the Kappa statistic for each of the five classifiers we used. Table 4 lists the percentage change of our accuracy, AUC, and kappa values from Alipour et al.'s metrics, while Table 5 lists the percentage change over Sun et al.'s metrics. Note that when Undef. is listed under the Sun et al. comparison table, it refers to the comparison between a positive value obtained by our study and a negative value from Sun et al.'s metrics.

Because our metrics showed a significant improvement over both those used by Alipour et al. and Sun et al., future studies in duplicate bug detection should consider including them. We then evaluated the information gain of each metric using Weka. This is detailed in Table 6.

The information gain of the metrics using the first shared topic to measure distance between topic-document distributions is significantly higher than that of metrics using Hellinger distance. Therefore, we hypothesize that this measure should be considered for use in a range of NLP applications. *simSumControl* has a high information gain as well,

Table 6: Information Gain of the Top Eight Metrics

sameTopicSum	0.330
sameTopicTot	0.321
topicSimSum	0.256
simSumControl	0.252
topicSimTot	0.209
sameTopicDes	0.203
topicSimDes	0.170
simDesControl	0.109

and may be a simple method with which to supplement or replace tf-idf methods in some text retrieval applications.

4. THREATS TO VALIDITY

The primary threat to construct validity concerns the number of bugs marked as duplicate in the Android database. A previous study showed an average of 36% of bug reports in repositories are duplicates or invalid [3], yet of the reports in the data, only 1,452 of 37,627 reports were marked as duplicate. In addition, some 10,000 reports in this time period have not been processed yet and are still marked as "New." Many of these could actually be duplicates. A threat to internal validity concerns the selection of pairs. Because more pairs are represented by large buckets of duplicates, there is a selection bias towards bug reports in larger groups rather than smaller ones.

5. CONCLUSION

The results of this paper demonstrate the descriptive power of the suite of attributes we have proposed. We have shown that they provide an increase in accuracy, AUC, and Kappa statistics over the metrics of Alipour et al. [2] and Sun et al. [11] when applied to the Android repository. However, our study simply extends the analysis by Alipour et al.: there is good reason to believe that the metrics used by Alipour et al., when combined with ours, could further improve duplicate bug detection.

The effectiveness and simplicity of the metrics proposed in this study make them good candidates for futures studies on duplicate bug reports. The practices of splitting the summary and description attributes when computing similarity, a strategy used by Jalbert and Weimar [6], and of using the first shared topic distance measure (which is, we believe, a novel distance measure) are particularly recommended. Finally, the use of Bagging to aid in classification should be considered, as it provided a small increase in accuracy.

A future study which utilized our metrics in order to find duplicates of incoming bugs, using a top-k approach similar to Sun et al. in which a fixed number of possible duplicate reports are presented for each incoming report, would further test their effectiveness. Our results suggest that this suite of metrics could significantly improve state-of-the-art top-k approaches to bug deduplication.

6. ACKNOWLEDGMENTS

We thank Alipour et al. for sharing the Android bug data used in their study. This material is based upon work supported by the U.S. Department of Education under Grant No. P200A100182 and by the National Science Foundation under Grant No. 1156563.

7. REFERENCES

[1] Stop words, 2013. http://www.link-assistant.com/seo-stop-words.html.

[2] A. Alipour, A. Hindle, and E. Stroulia. A contextual approach towards more accurate duplicate bug report detection. In *Proc. 10th Working Conf. on Mining Software Repositories*, pages 183–192, 2013.

[3] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proc. 28th Int'l Conf. on Software Engineering*, pages 361–370, 2006.

[4] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, Mar. 2003.

[5] G. Holmes, A. Donkin, and I. H. Witten. Weka: A machine learning workbench. In *Proc. 2nd Australian and New Zealand Conf. on Intelligent Information Systems*, pages 357–361, 1994.

[6] N. Jalbert and W. Weimer. Automated duplicate detection for bug tracking systems. In *Proc. IEEE Int'l Conf. on Dependable Systems and Networks*, pages 52–61, 2008.

[7] D. Lo, H. Cheng, and Lucia. Mining closed discriminative dyadic sequential patterns. In *Proc. 14th Int'l Conf. on Extending Database Technology*, pages 21–32, 2011.

[8] A. K. McCallum. MALLET: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[9] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proc. 27th IEEE/ACM Int'l Conf. on Automated Software Engineering*, pages 70–79, 2012.

[10] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[11] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang. Towards more accurate retrieval of duplicate bug reports. In *Proc. of the 26th IEEE/ACM Int'l Conf. on Automated Software Engineering*, pages 253–262, 2011.