Improving the Accuracy of Duplicate Bug Report Detection using Textual Similarity Measures

Alina Lazar, Sarah Ritchey, Bonita Sharif
Department of Computer Science and Information Systems
Youngstown State University
Youngstown, Ohio USA 44555
alazar@ysu.edu, sritchey@student.ysu.edu, bsharif@ysu.edu

ABSTRACT

The paper describes an improved method for automatic duplicate bug report detection based on new textual similarity features and binary classification. Using a set of new textual features, inspired from recent text similarity research, we train several binary classification models. A case study was conducted on three open source systems: Eclipse, Open Office, and Mozilla to determine the effectiveness of the improved method. A comparison is also made with current state-of-the-art approaches highlighting similarities and differences. Results indicate that the accuracy of the proposed method is better than previously reported research with respect to all three systems.

Categories and Subject Descriptors

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

General Terms

Management, Reliability

Keywords

Duplicate bug reports, textual similarity measures, support vector machines, TakeLab

1. INTRODUCTION

The complexity of software systems requires an automatic way to track bugs (and more generally change requests) discovered by users and developers. Bug tracking systems such as Bugzilla, include a search interface on the web that help users retrieve lists of bugs based on different criteria. However, often times, developers and users do not search to find if a bug has already been reported due to various time constraints. This leads to possible submissions of duplicate bug reports, referring to the same bug.

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/10.1145/2597073.2597088

During bug triaging, a triager looks at new bugs reported and assigns them to specific developers to be fixed. Before doing this, the triager also needs to manually determine if the newly reported bugs are duplicates of previously reported bugs. It would not be worthwhile for two developers to be fixing the exact same bug. This manual process is clearly time intensive as well as error prone.

In an attempt to solve this problem, many researchers [2, 4, 6, 8–11] have started looking into automatically detecting bug reports as duplicates of a previously reported bug. The methodologies and features used by previous researchers depend on one or more of the following: categorical feature analysis based on bug report fields, textual similarity features of the bug report's text using natural language processing techniques, stack traces, and analysis of the subject system's domain for contextual information. One of the most recent papers in this field by Alipour et al. [2] show that using contextual words extracted by LDA (an IR method) works better than textual similarity measures (e.g., BM25F) and categorical features combined.

In this paper, we present a new and improved method to determine textual similarity features. The textual similarity features are derived by *TakeLab* proposed by Saric et al. [7]. *TakeLab* is a set of two systems that automates measuring of semantic similarity of short texts using supervised machine learning. After all the features are calculated several binary classification methods including Naïve Bayes and Support Vector Machines are run to classify bugs as duplicate or non-duplicate. We report an improvement of 3.25% - 6.32% in accuracy across the three datasets tested. The main contribution of this paper is that the new textual features derived by *TakeLab* work well with classification methods to detect duplicate pairs of bugs and outperform previous work.

2. DATASETS USED

The three systems used in the case study presented here are Eclipse, Open Office, and Mozilla. Using web scraping techniques, bug reports were collected from the Bugzilla websites of the three systems. In Table 1 we report: the time interval when the bugs collected were submitted, the number of initial bugs collected, the number of initial duplicates, the number of bugs after preprocessing, and the number of duplicate pairs used for training.

After the bug reports were collected, reports that had an open resolution were removed from the datasets. We did this because their status cannot be confirmed from the information available. Removing them would help prevent training the model with mislabeled data. It is also important to note that this could easily be done in an industrial setting. The datasets were trimmed further by removing duplicate resolution bugs that did not have masters in the dataset.

For each dataset, the master duplicate bug was found together with all the duplicate bugs in it's group. This was just a preliminary step before generating all the duplicate bug report pairs. All these pairs are identified by adding to the dataset the classification decision feature *decision* and setting its value to 1. Four times as many non-duplicate pairs of bugs as duplicate pairs of bugs were generated for the training dataset. The decision for the non-duplicate pairs is set to -1.

These are the same datasets that are used in Sun et al. [8]. The Eclipse dataset is referred to as Eclipse2008 in Sun et al. However, the difference is that open bugs were not removed from their datasets. Our datasets do not contain open bug reports.

3. METHOD

This section describes our method for deriving the features and training our models. We first explain how the features are generated, followed by the classification method and the evaluation measures.

3.1 Generating Features

From over 20 bug properties downloaded, the following were selected as the base for the classification features: *bug_id*, *title*, *description*, *product*, *component*, *type*, *priority*, *version*, *open_date*. The textual properties, *title* and *description* are concatenated together and then used to generate 18 numeric features. These features, inspired by Saric et al. [7], are generated using the *simple* TakeLab system and include: n-gram word overlap for unigrams, bigrams and trigrams, n-gram word overlap for unigrams, bigrams and trigrams after lemmatization, WordNet based augmented word overlap, weighted word overlap, normalized differences for sentence length and aggregate word information content, shallow named entity and numbers overlap. The TakeLab system was designed to generate a similarity score between 1 and 5 for pairs of sentences or short text. After a set of features are calculated, Support Vector Regression predicts the similarity score. We refer the reader to [7] for more details.

All the other properties are input for the categorical measurement formulas listed below. Features 19 - 23 were adapted from Sun et. al's paper [8]. When calculating feature number 23 (version related), if the version is 'unspecified' for at least one of the bugs in the pair, the value of the feature is set to 0.5

$$feature_{19} = \begin{cases} 1, & \text{if } b_1.prod = b_2.prod \\ 0, & \text{otherwise} \end{cases}$$

$$feature_{20} = \begin{cases} 1, & \text{if } b_1.comp = b_2.comp \\ 0, & \text{otherwise} \end{cases}$$

$$feature_{21} = \begin{cases} 1, & \text{if } b_1.type = b_2.type \\ 0, & \text{otherwise} \end{cases}$$

$$feature_{22} = \frac{1}{1 - abs(b_1.priority - b_2.priority)}$$

$$feature_{23} = \frac{1}{1 - abs(b_1.version - b_2.version)},$$

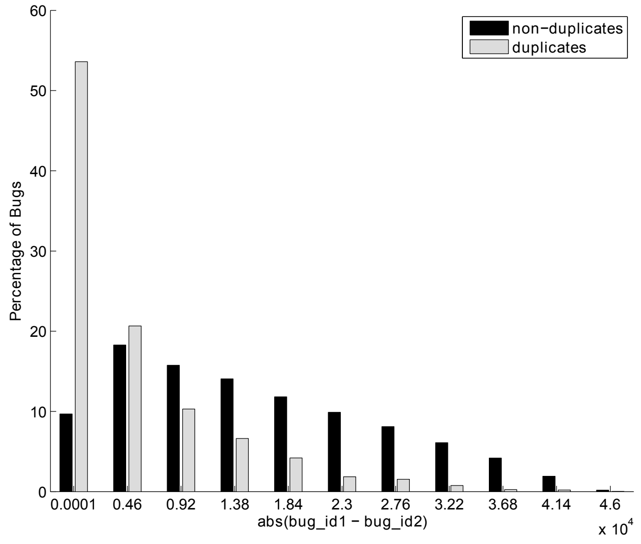

Figure 1: A histogram for feature 25. X-axis represents intervals for the bug_id differences

Runeson [6] concluded that 53% of all duplicates were submitted in an interval of 20 days after the master bug was submitted. Based on Runeson findings, we decided to calculate a new feature as the absolute value between the open dates of a pair of bugs.

$$feature_{24} = abs(b_1.open\_date - b_2.open\_date)$$

The last feature we used, inspired from Sureka et al. [10], computes the absolute difference between the bug_ids of the two bugs in the pair.

$$feature_{25} = abs(b_1.bug\_id - b_2.bug\_id)$$

This last feature is represented in Figure 1 as a group histogram that clearly shows the difference between the duplicate pairs as light gray and non-duplicate pairs represented as black. The feature discriminates the two categories well. Most duplicate bugs have a smaller bug_id difference compared with non-duplicate bugs.

3.2 Building the Training and Testing Sets

After we generate the features for all the bugs in the datasets, we divide the initial dataset into a training set and a testing set. The training set contains 5,000 bug pairs and all the other pairs are put into the testing set. The instances are divided into the subsets using stratified sampling, so the percentage between the two classes is preserved. That means out of the 5,000 pairs, 1,000 paris are duplicate and 4,000 are non-duplicate. Next, all the values in the training and the testing datasets are scaled to the [-1,1] interval. First the training set is scaled and the ranges are then applied to scale the testing set. The training sets are used to generate the classification models. For the SVM method, the two best parameters C and gamma are found performing a grid search done with cross-validation on the training set. After the model predicts the class for the each instance in the training set the evaluation measures are computed. All the data is available at `http://www.csis.ysu.edu/~alazar/msr14`.

Table 1: Details of Bug Datasets

Dataset	From	To	Original Bugs	Original Duplicates	Final Bugs	Final Duplicates	Duplicate Pairs
Eclipse	1/2008	12/2008	45,746	4,386	39,020	2,897	6,024
Open Office	1/2008	12/2010	31,333	4,549	23,108	2,861	6,945
Mozilla	1/2010	12/2010	78,236	10,777	65,941	6,534	16,631

Table 2: Eclipse Results

	Nearest Neighbors	Linear SVM	RBF SVM	Decision Tree	Random Forest	Naïve Bayes
Accuracy:	0.99992	0.99996	0.999841	0.999841	0.99996	1
Precision:	0.999605	0.999803	0.999211	0.999211	0.999803	1
Recall:	1	1	1	1	1	1
AUC:	0.99995	0.999975	0.9999	0.9999	0.999975	1

3.3 Binary Classification

At this point, the datasets contain pairs of duplicate and non-duplicate bug reports. To be able to automatically identify future bugs as duplicates of existing bugs, classification methods are used. The machine learning field provides several methods. One of the most popular algorithms today is the support vector machine and it's implementation called *LibSVM* [3]. The *SVM* classification model discriminates well between pairs of duplicate and non-duplicate bugs, provides excellent results in terms of accuracy and runs in an acceptable amount of time. There are two parameters that need to be fixed C and $gamma$. A grid search can successfully provide the best values for each dataset. Other classification methods are implemented in the Python package, *scikit-learn* [5]. We ran the following methods from *scikit-learn*: K-NN (K Nearest Neighbours), Linear SVM, RBF SVM, Decision Tree, Random Forest and Naïve Bayes.

3.4 Measures

Usually, classification methods are evaluated using the accuracy measure which is calculated as the percentage of correctly classified instances. However, the accuracy does not paint the entire picture, especially in case of unbalanced datasets. There are fewer duplicate bugs than non-duplicates in the datasets intrinsically. In the dataset we constructed, there were 4 times less pairs of duplicate bugs than non-duplicates. The accuracy can be still high, even if a significant number of instances from the positive class (duplicate pairs in our case) were classified incorrectly. To avoid this problem, we consider three other measures: precision, recall and the area under the curve (AUC). The standard definitions for these measures are used.

4. PRELIMINARY RESULTS AND OBSERVATIONS

In this section we show that the textual features extracted by TakeLab together with categorical features provide very good classification results. Accuracy is over 99% for all combinations of datasets and classification algorithms. A recall of 100% was obtained for all datasets. This means that all the positive instances (duplicate pairs of bugs) were correctly classified. Precision is also high, but not 100%. This means that sometimes few pairs of non-duplicate bug pairs were classified as duplicates. See Tables 2, 3, and 4 for the results of Eclipse, Open Office, and Mozilla respectively.

With respect to the Eclipse dataset, Naïve Bayes provides the best accuracy of 100%, which means that all the bug pairs were classified correctly. For Open Office, the highest accuracy is given by multiple algorithms, but recall is still 100%. From the confusion matrices we see only two pairs of bug reports were misclassified. The last table contains results related to the largest dataset in this case study: Mozilla. Linear SVM and Nearest Neighbors returned the best results and recall of 100%.

We also ran experiments using all 25 features, only TakeLab features (18) and only categorical features (7). All three experiments give almost identical results. We also experimented with the first three features from *TakeLab* and the five categorical features used by Sun et al. [8] and Alipour et al. [2] but this did not give good results.

The proposed approach is similar with the one described by Alipour et al. [2], with the exception of the features used. Three of the classification algorithms are common, but the additional algorithms we used may work better for larger datasets. The difference in results between the two approaches are reported in Table 5. The new set of textual features presented in this paper improves the accuracy between 3.25% and 6.32% over the contextual approach proposed by Alipour et al.

Sun et al. [8] were the first ones to propose the set of five categorical features (features 19 - 23), in addition of two textual measures based on the *BM25F* measure. The results in [8] are reported in terms of top-k recall rates and are not directly comparable with our results. However, no more than 80% of the duplicates were correctly identified compared with the 100% classification recall rates we obtained.

5. RELATED WORK

We present most relevant and recent work in the area of duplicate bug report detection. Runeson et al. first presented a way to determine if two bug reports were similar or possibly duplicates [6]. They used various types of natural language processing including stop word removal, stemming, and tokenizing to pre-process each report. The cosine similarity measurements between two documents were then used to compare similarities. Using this method, they were able to detect 40% of duplicate bug reports on a list of the most similar reports.

Sun et al. introduced a machine-learning model in 2010 [9]. It took into consideration different combinations of the summary and description of each report compared to another report. These combinations give 54 different features.

Table 3: Open Office Results

	Nearest Neighbors	Linear SVM	RBF SVM	Decision Tree	Random Forest	Naïve Bayes
Accuracy:	0.999933	0.999933	0.999697	0.999798	0.999899	0.999933
Precision:	0.99966	0.99966	0.998471	0.99898	0.99949	0.99966
Recall:	1	1	1	1	1	1
AUC:	0.999958	0.999958	0.999811	0.999874	0.999937	0.999958

Table 4: Mozilla Results

	Nearest Neighbors	Linear SVM	RBF SVM	Decision Tree	Random Forest	Naïve Bayes
Accuracy:	0.999949	0.999949	0.999936	0.999347	0.999936	0.999808
Precision:	0.999745	0.999745	0.999681	0.996943	0.999808	1
Recall:	1	1	1	0.999808	0.999872	0.999042
AUC:	0.999968	0.999968	0.99996	0.99952	0.999912	0.999521

Table 5: Comparison with Alipour's Results [2]

	New Features		Alipour	
	Accuracy	AUC	Accuracy	AUC
Eclipse	100.0000	1.0000	96.75	0.9900
Open Office	99.9899	0.9999	93.67	0.9660
Mozilla	99.9930	0.9999	94.78	0.9430

They had a 17-31% improvement for Open Office datasets, 22-26% improvement for Firefox datasets, and 35-43% improvement for Eclipse datasets relative to other state-of-the-art techniques. Next, they extended the BMF25 [8] by using bags of words and digrams to compare similarity in the title and description of each bug report. They used a stochastic gradient to then tune the parameters in their model to optimize the similarity measure. These changes resulted in a 10-27% relative improvement in recall rate and 17-23% relative improvement in mean average precision over their previous model.

Alipour et al. used architectural words from the Android system to organize the reports into feature and context topics by means of Latent Dirichlet Allocation (LDA) and labeled-LDA [1, 2]. They also implemented C4.5 and K-NN algorithms. This method produced results with 16.07% relative improvement in accuracy compared to [9].

Sureka et al. implemented character-based n-grams that are less susceptible to natural language related issues compared to word based systems [10]. This method requires little to no preprocessing. Once the data is augmented with a threshold, the system has a recall rate of 40.22% for top-10 size lists and 61.94% for top-50 size lists.

6. CONCLUSIONS AND FUTURE WORK

The paper presents an improved method to detect duplicate bug reports based on textual similarity measures. *TakeLab*, a text similarity system, is used to generate a majority of the features. A total of 25 new textual features are used. After determining the features, binary classification methods were run to categorize bugs into two classes: duplicate or non-duplicate. We tested this method on bug reports from Eclipse, Open Office, and Mozilla. Our method improves duplicate bug report detection by 6.32% even without the use of context based features as reported by Alipour et al. [2]. These preliminary results are very promising. In future work, we plan on using 10 times the size of the current datasets used to see if the current results hold.

7. REFERENCES

[1] A. Alipour. A Contextual Approach Towards More Accurate Duplicate Bug Report Detection . Master's thesis, University of Alberta, Canada, 2013.

[2] A. Alipour, A. Hindle, and E. Stroulia. A contextual approach towards more accurate duplicate bug report detection. *Proc. of the Tenth Intl Workshop on Mining Software Repositories*, pages 183–192, 2013.

[3] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Trans. on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[4] N. Jalbert and W. Weimer. Automated duplicate detection for bug tracking systems. *IEEE Intl Conf on Dependable Systems and Networks With FTCS and DCC*, pages 52–61, 2008.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[6] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. *29th ICSE*, pages 499–510, 2007.

[7] F. Saric, G. Glavas, M. Karan, J. Snajder, and B. Basic. Takelab: Systems for measuring semantic text similarity. In *Proc. of the First Joint Conference on Lexical and Computational Semantics*, pages 441–448, Montreal, Canada, June 2012.

[8] C. Sun, D. Lo, S. Khoo, and J. Jiang. Towards more accurate retrieval of duplicate bug reports. *Proc. of the 26th IEEE/ACM ASE*, pages 253–262, 2011.

[9] C. Sun, D. Lo, X. Wang, J. Jiang, and S. Khoo. A discriminative model approach for accurate duplicate bug report retrieval. *Proc. 32nd ICSE*, 1:45–54, 2010.

[10] A. Sureka and P. Jalote. Detecting duplicate bug report using character n-gram-based features. *APSEC*, pages 366–374, 2010.

[11] Y. Tian, C. Sun, and D. Lo. Improved duplicate bug report identification. *16th European CSMR*, pages 385–390, 2012.