FourD: Do Developers Discuss Design? Revisited

Abbas Shakiba Robert Green Robert Dyer
Bowling Green State University
Bowling Green, OH, USA
{sshakib,greenr,rdyer}@bgsu.edu

ABSTRACT

Software repositories contain a variety of information that can be mined and utilized to enhance software engineering processes. Patterns stored in software repository meta-data can provide useful and informative information about different aspects of a project, particularly those that may not be obvious for developers. One such aspect is the role of software design in a project. The messages connected to each commit in the repository note not only what changes have been made to project files, but potentially if those changes have somehow manipulated the design of the software.

In this paper, a sample of commit messages from a random sample of projects on GitHub and SourceForge are manually classified as "design" or "non-design" based on a survey. The resulting data is then used to train multiple machine learning algorithms in order to determine if it is possible to predict whether or not a single commit is discussing software design. Our results show the Random Forest classifier performed best on our combined data set with a G-mean of 75.01.

CCS Concepts

•Software and its engineering → Software libraries and repositories;

Keywords

mining; software design; Boa; machine-learning

1. INTRODUCTION

As is well known, software repositories hold a great amount of information about projects and their structures (meta data). Some of this information is easily measurable and some is not, often being hidden in discussions, commit messages, etc. In particular, commits hold a surprisingly large amount of information as they often contain discussions between developers that may contain hidden patterns within a project. Topics of discussion in these messages can show trends of the project and which subjects developers are more interested in or are struggling with.

In this paper, software repositories are mined using multiple machine learning algorithms to find out how many of the commits are discussing the design of the software system. As is noted in many studies [1, 2, 4, 9], various aspects of design are of significant importance to the software engineering community. To the best of the authors' knowledge, this task has only been previously undertaken by a single work [2]. Though presenting a clear methodology and significant results, the work presented multiple opportunities for extension. First, the study included data from a single hosting service, GitHub, leaving room for the evaluation of data from other hosting platforms. Second, the study used only two classification techniques, leaving room for the application of new techniques to achieve improved results. Third, the study was not truly repeatable as the data set used was not provided.

Based on this, this work extends the state-of-the-art through multiple contributions including: 1) the use of meta-data from multiple software repository hosting platforms (SourceForge and GitHub) and 2) the inclusion of additional computational intelligence techniques including random forests (RF), decision trees (DT), naive Bayes classification (NB), k-nearest neighbor (KNN), multinomial Bayes classification (MB), and support vector machines (SVMs).

The initial mining to extract commit information is completed using the Boa language and infrastructure [5]. The resultant data is then manually classified (by the authors and through a survey) in order to tag commits as discussing software design or not. Based on this manually classified data, machine learning is applied to answer the question, "can we automate finding commits referencing design?"

Our results show both the SVM and RF classifiers perform well on this data, while RF has the highest overall G_{mean}. For many techniques the accuracy rate was over 80%, indicating that automatically classifying the commits discussing design is realistic. Such automation could allow software designers to quickly see what discussion is occurring and the related code changes surrounding that discussion.

2. METHODOLOGY

This study uses several platforms and programming languages to complete the presented research. Boa [5] was used to extract and collect raw data from two very large scale software repositories (GitHub and SourceForge). An online survey tool written in Ruby on Rails was also created to collect data from various contributors who manually classified

SWAN'16, November 13, 2016, Seattle, WA, USA

http://dx.doi.org/10.1145/2989238.2989244

```
1 PROJECTS: output top(5) of string weight float;

2 isempty := function(s: string) : bool {
3   if (match(`^\s*|no message$`, s)) return true;
4   if (s == "*** empty log message ***") return true;
5   return false;
6 };

7 count := 0;

8 visit(input, visitor {
9   before CodeRepository -> count = 0;
10  after CodeRepository -> if (count >= 200)
11    PROJECTS << input.id weight rand();
12  before rev: Revision ->
13    if (!isempty(trim(lowercase(rev.log))))
14      count++;
15 });
```

Figure 1: Boa code to randomly select 5 projects with at least 200 non-empty commits.

```
1 COMMITS: output top(200)[string] of string weight float;

2 ids := {
3   "6176545", "6150849", "209281", "13151128", "1019785"
4 };

5 exists (i: int; input.id == ids[i])
6   visit(input, visitor {
7     before rev: Revision ->
8       if (!isempty(trim(lowercase(rev.log))))
9         COMMITS[input.id] << rev.log weight rand();
10  });
```

Figure 2: Boa code to randomly select 200 commits from the 5 randomly selected projects in GitHub.

our data sets. Weka, a machine learning and data analysis software written in Java [6], was also used to build our prediction models based on different classification algorithms. The remainder of this section describes each of these steps.

2.1 Step 1: Data Extraction

Boa is a domain specific language for mining large-scale software repositories. The goal of Boa is to ease testing hypotheses and questions related to mining software repositories. This study uses Boa to extract 200 random commits from five randomly selected projects in multiple software repository (five from GitHub and five from SourceForge) resulting in a total sample of 2,000 commits.

Figure 1 shows the Boa query to randomly select 5 projects from the analyzed data set. Lines 2–6 and 12–13 are simple pre-processing instructions to check and remove empty commits. Lines 9–10 ensure only projects with at least 200 non-empty commits are included. A top-5 output aggregator (line 1) is used to select 5 projects. The project IDs are sent to the aggregator (line 11) with random weights ensuring a random sample.

Figure 2 shows the Boa query used to to extract commits from each software repository. Lines 7–10 ensure the query only analyzes the randomly selected projects from the previous query. A top-200 output aggregator (line 1) is used to select 200 commits from each selected project. The commit logs are sent to the aggregator (line 14) with random weights and are indexed by project ID, ensuring a random selection of 200 commits per project.

Table 1: WEKA Pre-processing configurations.

StringToWordVector	
`IDFTransform`: true	`lowerCaseTokens`: true
`stemmer`: IteratedLovinsStemmer	
`tokenizer`: AlphabeticTokenizer	

2.2 Step 2: Label Collection

In the second step, a website was developed to manually classify commits as discussing software design or not. The survey was web-based so that software experts could vote for each commit and whether or not the commit discusses design or not. The web application was written in Ruby on Rails, leveraged MySQL for data storage, and used Bootstrap to ease the front-end design.

For each commit, three options were presented to each survey participant: 1) discusses design; 2) does not discuss design; and 3) can not decide. Based on the majority vote for each commit (with 2 votes minimum), the commit was classified as *design* or *non-design*. The final results of the manual classification of the data set showed that only 14% of commits discuss design including 139 out of 1,000 from GitHub's projects (13.9%) and 140 out of 1,000 from SourceForge's projects (14%). This distribution may vary based on which projects are randomly selected.

2.3 Step 3: Pre-processing

In the last step, the collected data from the online survey was converted to a Weka input file in the *arff* format. Weka is software that contains data analysis, predictive modeling, and visualization tools to aid in the application of machine learning techniques [7]. Weka provides a simple graphical user interface so that users can work with different machine learning algorithms, filters, and change the configurations of algorithms easily without having any deep knowledge of Java programming [7].

The data was pre-processed by removing short and common words, applying a word-stemming algorithm, alphabetically tokenizing words, and then ranking all remaining words (see Table 1 for WEKA settings).

3. EVALUATION

For evaluating the proposed methodology, the data collected was split into three data sets. The first is GitHub-only data, the second is SourceForge-only data, and the third is a combination of both data sets. The data sets were split between GitHub and SourceForge for comparative analysis to answer the questions, "do GitHub or SourceForge developers speak about design more often?" Multiple classification algorithms were applied to each data set using 10-fold validation. Multiple experiments were also run using different weights to encourage improved results. As was mentioned in Section 2.2, only 14% of all commits discuss design which makes this data set an imbalanced one, which may degrade performance of the classifiers [8].

In an imbalanced data set, one approach to increase the accuracy is to add extra cost to the false negatives of the minority class [3]. In this paper's first scenario, all costs were balanced. In the second scenario, costs were adjusted

Table 2: Results for Decision Tree.

Decision Tree					
Data	Acc.	%TP	%TN	% F-M	G-mean
GitHub	86.18	98.3	12.2	82.8	68.50
sForge	87.39	97.6	25	84.7	74.81
Both	86.99	96.5	20.8	85.2	65.82
Increased False Negative Weight					
GitHub	82.98	90.2	38.1	82.9	58.94
sForge	82.58	87.3	53.6	83.5	61.26
Both	82.13	87.8	47.3	69.70	59.32

Table 3: Results for Random Forest.

Random Forest					
Data	Acc.	%TP	%TN	%F-M	G-mean
GitHub	86.38	99.0	8.6	86.4	71.14
sForge	86.98	99.8	8.5	82.0	87.23
Both	86.44	99.5	5.7	81.4	75.01
Increased False Negative Weight					
GitHub	87.58	96.4	33.1	86.0	73.31
sForge	87.49	95.7	37.1	86.3	72.68
Both	87.78	96.5	34.0	86.3	74.21

Table 4: Results for Naive Bayes.

Naive Bayes					
Data	Acc.	%TP	%TN	%F-M	G-mean
GitHub	75.17	76.2	69	78.6	54.72
sForge	81.48	84.2	65	83.2	61.33
Both	79.63	82.4	62.3	81.4	58.28
Increased False Negative Weight					
GitHub	69.70	68.8	74.9	74.2	51.38
sForge	75.18	74.5	79.3	78.6	56.73
Both	67.01	65.3	77.4	72.0	50.17

to encourage learning imbalanced data to decrease the false negative rate of the minority class.

For each classification algorithm used, the parameters were set as listed in Table 1. For pre-processing, the *StringWordtoVector* pre-filter is used with the the *IteratedLovinsStemmer* stemmer and *AlphabeticTokenizer*. Also, values of *IDFTransform*, *FTRansform*, and *lowerCaseTokensare* are set to *true*. The results of this analysis are shown in Tables 2–7 and Figure 3.

Each table containing results has five columns and two sections where *ACC* stands for accuracy, *%TP* stands for true positive rate, *%TN* stands for true negative rate, *%F-M* stands for f-measure rate, and *G-Mean* stands for geometric mean. As the data is imbalanced, the accuracy of classification is not necessarily a proper and accurate way to measure the performance of the classification. For example, in an extremely imbalanced data set, a high rate of classification can occur without classifying the minority class correctly. To evaluate performance of classification there are other metrics like True Negative, True Positive, F-Measure, G-mean, etc. Those metrics can evaluate the performance of learning imbalanced data [3]. Each result table has two sections. In the top section, the entire confusion matrix has the same weight, but in the bottom section the weight of false negative cells was increased to encourage the classifier to obtain a higher true positive for the minority class according to Table 8.

Table 5: Results for Multinomial Bayes.

Multinomial Bayes					
Data	Acc.	%TP	%TN	%F-M	G-mean
GitHub	74.27	74.8	71.2	78.0	54.33
sForge	70.57	70.0	73.5	85.5	51.85
Both	4.32	74.7	72.0	87.0	54.57
Increased False Negative Weight					
GitHub	81.28	86.7	47.5	82.3	57.74
sForge	78.98	82.9	55.0	80.9	56.23
Both	83.74	86.4	67.4	85.0	64.81

Table 6: Results for Support Vector Machine.

Support Vector Machine					
Data	Acc.	%TP	%TN	%F-M	G-mean
GitHub	85.48	93.7	34.5	84.7	64.95
sForge	86.27	93.7	40.7	85.7	68.21
Both	86.38	95.3	31.2	84.9	68.14
Increased False Negative Weight					
GitHub	85.58	92.1	45.3	85.4	66.25
sForge	85.69	92.0	47.1	85.6	72.68
Both	85.34	90.1	50.8	85.5	64.61

Table 7: Results for Support K-Nearest neighbor.

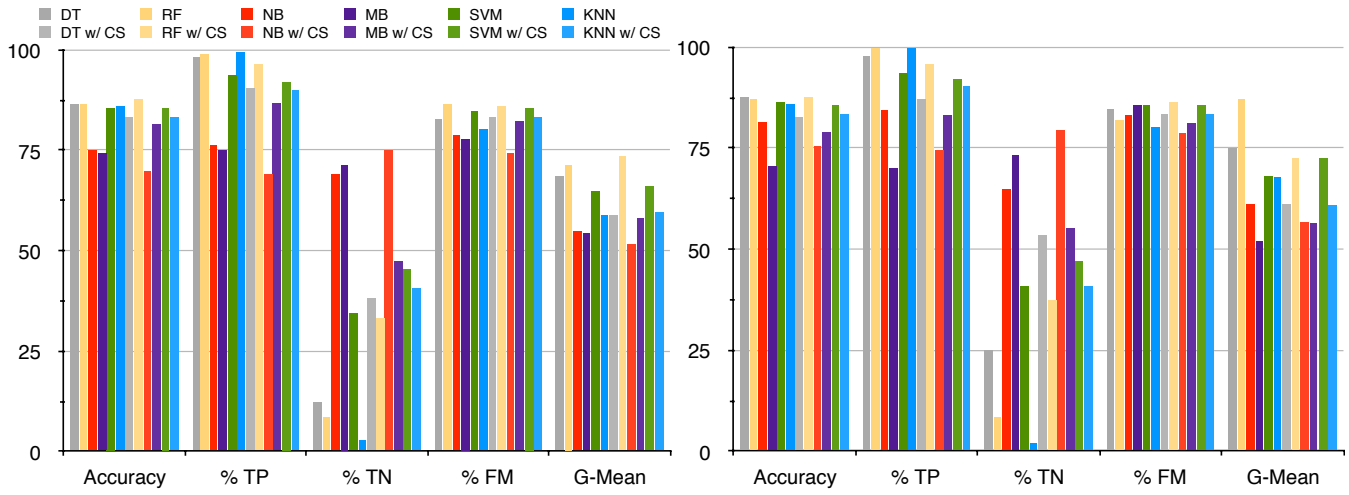
K-nearest neighbor					
Data	Acc.	%TP	%TN	%F-M	G-mean
GitHub	85.89	99.3	2.9	80.3	58.85
sForge	85.99	99.7	2.1	80.1	67.81
Both	86.18	99.9	1.8	80.1	80.16
Increased False Negative Weight					
GitHub	82.89	89.9	40.3	83.0	59.50
sForge	83.38	90.3	40.7	83.4	60.59
Both	85.43	95.6	21.9	83.1	62.81

Table 8: Execution configurations (using WEKA defaults).

Algorithm	Confusion Matrix
Decision Tree	NA
Decision Tree w/ CS	[[0, 1], [10, 1]]
Random Forest	NA
Random Forest w/ CS	[[0, 1], [8, 1]]
Naive Bayes	NA
Naive Bayes w/ CS	[[0, 1], [8, 1]]
M-Bayes	NA
M-Bayes w/ CS	[[0, 1], [4, 0]]
SVN	NA
SVN w/ CS	[[0, 1], [12, 0]]
KNN	NA
KNN w/ CS (with K = 2)	[[0, 1], [17, 0]]

4. DISCUSSION

In evaluating the performance of these methods, it is important to decide what criteria is the most important. For example, if only the overall accuracy rate is important, K-Nearest Neighbor would be a quick solution to answer the question. Although the accuracy rate of this algorithm is the highest among others, this algorithm misclassified almost all of those commits referring to design. At the other end of the spectrum, the Naive Bayes method shows a reasonable



(a) GitHub data set. (b) SourceForge data set.

Figure 3: The results of all classification algorithms.

balance between accuracy and false-positive rate, suggesting it is the superior method.

One solution to evaluate these methods more thoroughly and fairly is to use a statistical solution. As this study deals with imbalanced data, metrics like F-measure, G-mean, etc. are most applicable. Kim *et al.* [8] came up with some approaches to resolve problems caused by imbalanced data. One of these approaches is using G_{mean} as a statistical variable to help select between results. The calculation of G_{mean} is shown in (1).

$$G_{mean} = \sqrt{\frac{TP}{TP+FN} * \frac{TN}{TP+FP}} \quad (1)$$

A higher value of G_{mean} suggests a superior result. Using this statistic, the best performing algorithms were SVM for GitHub, Random Forest for SourceForge, and again Random Forest for the complete data set. Thus overall, Random Forest performs the best.

5. CONCLUSIONS AND FUTURE WORK

This paper revisited the question of "Do Developers Discuss Design?" using data mined via Boa to expand the analysis of this key question to repository meta-data across multiple repository sites. Further, new sets of classification methods have also been explored. Results using the G_{mean} statistic demonstrate that both SVM and Random Forest methods work well, but the Random Forest method appears to be superior with an overall G_{mean} of 75.01.

We are currently working to integrate the machine learning techniques into Boa. This would allow us to train the classifiers inside Boa and then easily use those classifiers on a large set of data, thus proving the feasibility of automatically detecting commits discussing design. In the future, this work may also be extended to include larger data sets, more classification methods, ensemble based-methods.

6. ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation under CCF-15-18776 and CNS-15-12947.

7. REFERENCES

[1] S. Amann, S. Beyer, K. Kevic, and H. Gall. *Software Engineering: International Summer Schools, LASER 2013-2014, Revised Tutorial Lectures*, chapter Software Mining Studies: Goals, Approaches, Artifacts, and Replicability, pages 121–158. Springer International Publishing, Cham, 2015.

[2] J. Brunet, G. C. Murphy, R. Terra, J. Figueiredo, and D. Serey. Do developers discuss design? In *11th Working Conference on Mining Software Repositories*, MSR, pages 340–343, 2014.

[3] C. Chen, A. Liaw, and L. Breiman. Using random forest to learn imbalanced data. *University of California, Berkeley*, 2004.

[4] W. Ding, P. Liang, A. Tang, H. Van Vliet, and M. Shahin. How do open source communities document software architecture: An exploratory survey. In *International Conference on Engineering of Complex Computer Systems*, pages 136–145, Aug. 2014.

[5] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *35th International Conference on Software Engineering*, ICSE, pages 422–431, 2013.

[6] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

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

[8] M.-J. Kim, D.-K. Kang, and H. B. Kim. Geometric mean based boosting algorithm with over-sampling to resolve data imbalance problem for bankruptcy prediction. *Expert Systems with Applications*, 42(3):1074–1082, 2015.

[9] H. Unphon and Y. Dittrich. Software architecture awareness in long-term software product evolution. *J. Syst. Softw.*, 83(11):2211–2226, November 2010.