Where is the Road for Issue Reports Classification Based on Text Mining?

Qiang Fan, Yue Yu*, Gang Yin, Tao Wang, Huaimin Wang
National Laboratory for Parallel and Distributed Processing
College of Computer, National University of Defence Technology
Changsha, China
{fanqiang09, yuyue, yingang, taowang2005, hmwang}@nudt.edu.cn

***Abstract*—Currently, open source projects receive various kinds of issues daily, because of the extreme openness of Issue Tracking System (ITS) in GitHub. ITS is a labor-intensive and time-consuming task of issue categorization for project managers. However, a contributor is only required a short textual abstract to report an issue in GitHub. Thus, most traditional classification approaches based on detailed and structured data (*e.g.*, priority, severity, software version and so on) are difficult to adopt.**

In this paper, issue classification approaches on a large-scale dataset, including 80 popular projects and over 252,000 issue reports collected from GitHub, were investigated. First, four traditional text-based classification methods and their performances were discussed. Semantic perplexity (*i.e.*, an issues description confuses bug-related sentences with nonbug-related sentences) is a crucial factor that affects the classification performances based on quantitative and qualitative study. Finally, A two-stage classifier framework based on the novel metrics of semantic perplexity of issue reports was designed. Results show that our two-stage classification can significantly improve issue classification performances.

***Index Terms*—issue tracking system; machine learning technique; mining software repositories;**

I. Introduction

The prosperity of open source software (OSS) resulted in an increasing number of developers joining in and contributing to OSS communities. GitHub is one of the most popular social coding communities that attracts a large number of developers [1] . As of April 2016[1], over 14 million registered users are collaborating in GitHub, showing a great power in driving the OSS project forward. Based on the perspective of contributors, reporting issues using issue tracking system (ITS) is one of the most important activities in OSS communities [2]. Section II-A describes that the GitHub-provided ITS is more lightweight to use today, compared to the traditional ITS (e.g., Bugzilla). A contributor is only required a short textual abstract, containing a title and an optional description, to report a new issue in GitHub. Therefore, this simplified process of reporting issues decreases the barrier to entry and attracts more inexperienced external contributors. According to our statistics, Ruby on Rails, one of the most active projects in GitHub, receives upwards of 700 new issues each month.

However, the extreme openness of ITS poses a serious challenge for the core team in project maintenance. In large-scale projects, many undesirable and vague issue reports are submitted by external contributors (*e.g.*, asking questions, as shown in Figure 1) because of their reluctance to spend adequate time to read and comprehend the contribution guidelines (as shown in Figure 2), which provide details on reporting a high-quality issue and the kind of issue the project prefers. Thus, issue categorization is a labor-intensive and time-consuming task for project managers. Furthermore, the core team members have to provide quick responses and resolve the incoming issues in time to sustain the passion of external contributors [3].

Fig. 1. Example of undesirable issue in ITS

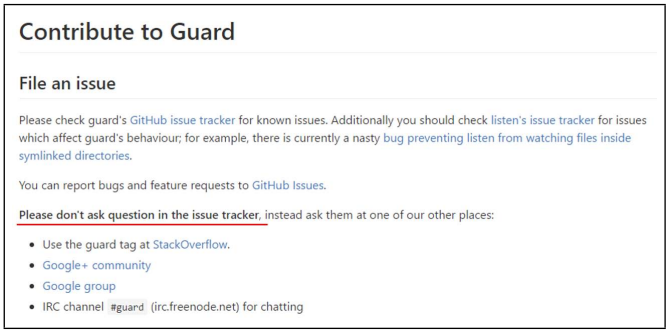

Fig. 2. Example of contribution guidelines in GitHub

Most issue management tasks are organized based on the label system in GitHub[2] (as discussed in Section II-B). One of the most popular practices is distinguishing different types [4] of issues (*e.g.*, bug, feature request, and refactoring), which is a manual process maintained by core developers. Thus, the

*Corresponding author.

[1]https://en.wikipedia.org/wiki/GitHub

[2]https://help.github.com/articles/applying-labels-to-issues-and-pull-requests/

high performance of issue categorizing approach, especially issues established in limited prior information (*i.e.*, the majority of issues only have textual summary and historical data of submitters), could significantly reduce the cost of issue management.

This paper focuses on the challenge of distinguishing real bugs from nonbugs among all issues, similar to prior work [5], [6]. Although a fine-grained classification is deferred for future work, we argue that this work can 1) greatly improve the efficiency of issue management in GitHub because well-established projects prefer to maintain limited types of issues, especially for bug and feature (*e.g.*, Ruby on Rails[3], and angualr.js[4]); 2) reduce the noise and bias [5], [7] introduced by confusing real bugs with other types of issue (*e.g.*, feature requests), when building bug prediction [8], [9] or software quality [10] models based on mining the big data from GitHub.

In summary, the key contributions of this paper include:

- The study of text-based classification approaches on a large-scale dataset. Four different machine learning classifiers were evaluated on 80 popular projects in GitHub. The results show that the support vector machines (SVM) achieve the best performance.
- The limitations of text-based classification approaches were analyzed, and the results showed that semantic perplexity (*i.e.*, an issue's description confuses bug-related sentences with nonbug-related sentences) is a crucial factor that affects classification performances. Thus, representative metrics were designed to quantify the semantic perplexity of an issue report.
- A novel two-stage classifying framework was designed to improve the performances of traditional classification models. Features relating to semantic perplexity were extracted from free text in the first stage, and then a synthesized classification model was built in the second stage. The quantitative evaluations show that classification performance can achieve a significant improvement.

The structure of this paper is organized as follows. In Section 2, we introduce the background of our study, and illustrate related work and our research questions. In section 3, we present the key process of building effective classification models. The results and discussion can be found in Section 4. Finally, we draw our conclusions in Section 5.

II. Background and Related Work

A. Issue Tracking System

Software development generally produces programs with two caveats [11]: (1) they are often incomplete with respect to certain features, and (2) they are usually buggy. In the development process, most of time, developers code and test the programs, while end-users (does not exclude developers) use the programs and provide feedbacks. Both developers and end-users can submit issues to ITS when the software performance does not meet their expectations. Then, the core team needs to clearly understand the intentions of the contributors, distinguish categories, and find suitable developers to fix the corresponding issues at the stage of project maintenance. Using the ITS is a common way to organize and maintain development tasks in the open source practice [12] to help project mangers keep track of issue reports by monitoring progress, identifying new issues, discussing potential solutions for fixing bugs, and so on. The consistent utilization of ITS is considered as a "hallmarks of a good software team" [13] in open source communities.

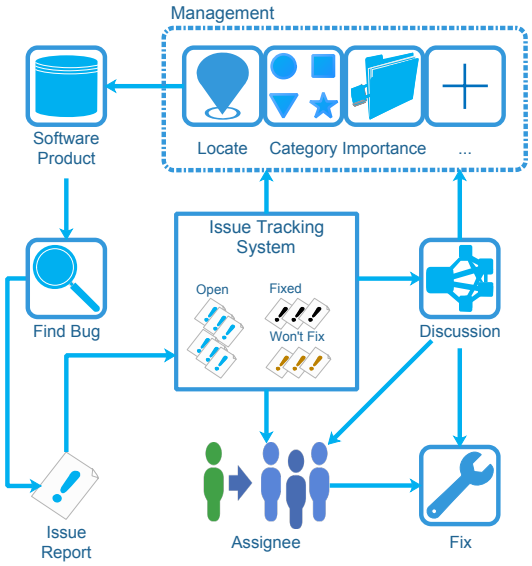

Fig. 3. Workflow of management for traditional ITS

Dozens of ITS tools has been popularized, *e.g.*, *Bugzilla* and *ITracker*, with the development of OSS. These traditional tools design a rigid and complicated data structure, which are used for organizing issues, *e.g.*, category, priority, assignee and status. Figure 3 shows the common workflow. First, when contributors find bugs in the software, they are usually asked for a basic description about the bug, and the structured fields are completed to as many as they can. Second, core team members and contributors discuss their problems to clearly understand the problems. Thereafter, the structured information that requires distinguishing suitable categories, determining priorities, making plans to ensure progress, and locating the issue (indicating the product, component, and version of the software where the issue appeared) are corrected. Finally, based on all structured information, the corresponding developers would be assigned to fix the bugs.

Several academic studies focus on ITS to free the managers from some cumbersome and repetitive work, such as automatically classifying issue reports to bug-prone and nonbug-prone [4]–[6], bug assignment [14], [15], duplicate issue detection [16], [17], fixing time prediction [18], and so on. Most of the existing approaches highly depend on the structured bug data (*e.g.*, priority and severity). However, prior work [4]–[6] has shown that OSS contributors often omit or use default

[3]http://edgeguides.rubyonrails.org/contributing_to_ruby_on_rails.html

[4]https://github.com/angular/angular/blob/master/CONTRIBUTING.md#issue

value for some important information, which results in many wrong messages and missing messages existing in ITS. Thus, improving the efficiency of ITS services is becoming an important research topic [12].

B. Lightweight ITS in GitHub

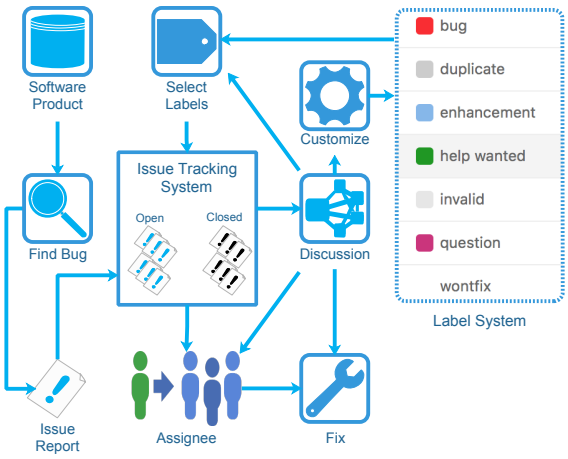

Fig. 4. Workflow of management in GitHub ITS

GitHub, the largest social coding community, released its own ITS called *Issues 2.0*[5] in 2014 to provide an excellent service in reporting issues. Figure 4 summarizes the typical workflow of issue management in GitHub. First, the contributor submits an issue report and provides some textual summary to describe it. Second, the core team of the project discusses the issue with the contributor. During discussion, the core team needs to reach an agreement of the issue with the contributors, and select relevant labels for the issue from pre-defined labels. Finally, the core team assigns the issue to be fixed by the corresponding developer.

GitHub provides a more lightweight ITS that is flexibly integrated with its label system, compared to traditional ITS. The structured information of issues, such as category and priority, is substituted by the label system in GitHub. Contributors are only required a short textual abstract when submit issues, whereas the core team can use labels, besides milestone and assignee, to mark and manage issue reports. The label system in GitHub is custom and the core team can summarize information concerns them most as labels. Zach Holman, a GitHub engineer, describes his design as follows: *"Our goal is basically to make a flexible, simple product that everyone can enjoy." "In the meantime, we've found that using labels and milestones is a great way to achieve the same result (other functions) in a more flexible system."*

The lightweight design of ITS results in some changes for the contributors and core team. For contributors, Issue 2.0 reduced the cost of submitting issue reports and stimulated their enthusiasm. However, its openness results in the emergence of undesired issues in ITS. Consequently, loose constraints reduced the work of contributors, which are transferred to the core team as management task. Thus, an automatic approach that can effectively filter out useful issue reports is significant and urgent for issue management. For the core team, the flexible label system makes the management task customizable and configurable. The flexibility breaks the fixed form of management, and the core team can shape the way of management according to their requirement. However, this customization lacks mandate and enforceability, thereby omitting structured information. Moreover, this customization results in difference in usage among projects, which makes the management of different projects difficult to understand.

A study on a large-scale dataset was performed to build a common and effective text-based classification approach for GitHub projects. This paper focuses on classifying bug-prone and non-bug prone issue reports because of the dominance of bugs in ITS. We expect to refer the achievement from former research on traditional ITS, so we ask:

RQ1: ***Do traditional text-based classification approaches still work on the ITS of GitHub? Which classifier performs best?***

In prior research [6], [19], combining textual and structured information (*e.g.*, priority, assignee, and so on) is used routinely to outperform the classifier. However, as previously discussed, the omission of structured information is serious in the ITS of GitHub. The data that can be used to build a classifier are limited because of the structured information scarcity. Thus, only textual summary and the historical data of submitters can be used for the majority of issues. Facing this challenge, we expect to study some factors that may influence the performance of classifiers, so we ask:

RQ2: ***What factors influence the performances of text-based classification approaches?***

The factors that influence the performances of text-based classifiers, which would guide us to extract more additional features from textual summary, can be identified using the regression analysis. In this paper, a two-stage classifier framework, which can flexibly combine textual information and other types of features, was built. To evaluate our approach, we ask:

RQ3: ***How to improve the classification performances by integrating different types of features, especially for the semantic complexity metrics extracted from textual descriptions?***

C. Related Work

Many studies have investigated bug classification [4]–[6] to predict whether an issue is about a bug.

Antoniol et al. [4] investigated the automatic classification of issue reports by utilizing conventional text mining techniques based on the description part of issue reports. By extracting the textual part of issue reports (title, description, and discussions) from ITS of three case projects and building classifiers using three supervised MLTs (alternating decision trees, Naive Bayes classifiers, and logistic regression), linguistic information in ITS is sufficient (82% best precision

[5]https://github.com/blog/831-issues-2-0-the-next-generation

for three case projects) to automatically distinguish bugs from other activities.

Zhou et al. [6] proposed a hybrid approach that combines text and data mining techniques and considers the misclassification of issue reports in ITS. They took advantage of structural information with textual information that proposed a hybrid approach that combines text and data mining techniques, and achieved an excellent result (average 84.7 for 5 case projects).

A common approach adds information extracted from ITS [4], [6], [19] to improve the performance of the model. In [4], discussions are involved; in [6], structural information, such as severity, priority, and component, are utilized; and in [19], metadata extracted from ITS are proved to outperform classifiers. However, these kinds of data are not produced when the issue report is submitted. Consequently, obtaining structured information in GitHub is difficult because of omissions. Textual summary is a main type of information used to build the classification model.

III. DATA SET

A. Data Collection

The dataset from Yu [20], which is a comprehensive dataset to study the pull-based model, involving 951,918 issues across 1,185 main-line projects in GitHub (dump dated 10/11/2014 based on GHTorrent [21], [22]), was used in this paper. Data on title, content, labels, and contributors for each issue were obtained through GitHub API. Projects need to contain a sufficient number of labeled issues for training and testing to test which supervised text-based classification performs best. Otherwise, an appropriative number of bug and non-bug issues existing in the projects is required to avoid the influence of unbalanced dataset. Thus, the candidate projects from GitHub, which have at least 500 labeled issues and bug rate between 20% and 80% (the labeling dataset process is presented in section III-B), were finally identified and can be used as the training and testing sets.

B. Category Extraction

Compared to traditional ITS, the ITS in GitHub uses a labeling system to manage issues. Category information from the user-defined label system must be extracted to obtain a pre-labeled training set from GitHub. Our dataset has 7,793 different labels in 1,185 projects, and many projects use different tags (*i.e.*, labels), such as bug, type:bug, error, and defect to express the same meaning and identify bug-related issues. A qualitative study is presented in this paper to comprehend the use of tags as categories of issues in this section.

Many projects in GitHub provide additional information in tags. For example, in project "WordPress-Android", issues are labeled as "[type] bug", "[type] enhancement", and so on. Tags in these projects contain not only categories of issues but also categories of the tag itself. The information is useful in knowing what labels are mostly used to express categories of issues.

A process is designed to aggregate these tags by fully utilizing the additional information. First, all tags that act as those forms were selected, and their information were separated. A 2D vector $< C, name >$ was used to represent these tags, where C is the category of the tag (such as "type", "component") and $name$ is the main information of the tag (such as "bug", "feature"). Second, tags with similar C items were grouped as $Group_C$. Then, the similarities of the two groups were defined using Equation 1. The groups whose similarity were greater than the threshold were merged.

$$similarity = \frac{\left|Group_{C_i} \bigcap Group_{C_j}\right|}{\min\left(\left|GroupC_i\right|, \left|GroupC_j\right|\right)}, (i \neq j) \quad (1)$$

$Group_{C_i}$ is a set of tags with the same category C_i and different $name$. Finally, a structure tag information is obtained through the aforementioned process.

TABLE I
TAGS IN GITHUB

Category	Label	Projects	Issues	Percent	Total
bug	bug	644	118,155	46.9%	52.2%
	defect	15	7,604	3.0%	
	type:bug	13	5,684	2.3%	
nonbug	enhancement	412	44,947	17.8%	38.6%
	feature	199	14,795	5.9%	
	question	319	13,109	5.2%	
	feature request	93	6,976	2.8%	
	documentation	239	6,422	2.5%	
	improvement	45	5,592	2.2%	
	docs	122	5,510	2.2%	

Through the prior process, we extract 149 tags, which can indicate the category of issues, as group "$type$". Finally, we filter total 252,084 issues with tags in group group "$type$". Table I shows the most used tags in group "$type$", and how many projects and issues they appear. These tags were divided into bug-prone or nonbug-prone by manually distinguishing. The most used tags are bug, enhancement, and feature, which were observed in 46.9%, 17.8%, and 5.9% of the labeled issues, respectively. The issues with other tags These structure tags were used in this study to judge whether an issue is bug-prone or not.

The number of issues labeled with both categories is 3,869 in 386 projects, which can be ignored compared with 252,084 labeled issues.

C. Preprocessing of Dataset

Each issue, which can be labeled as "bug" or "non-bug", is characterized by its title and description. In this paper, issues labeled by the former process were selected to perform the following steps. First, linguistic features extracted for the text-based classifier undergo standard processing, i.e., lowercase, text filtering, stemming, and indexing [23]. All stop-words and common English terms, such as "should", "might", "not", were retained. The importance of linguistic features for classifying issues was indicated in study [4]; moreover, study [11] mentions that removing the default list of stop-words in common

corpora might decrease the classification accuracy. Otherwise, "```" is used to distinguish the code information in issue reports, because markdown editor is used in GitHub.

Then, a vector space model was used to represent each issue as a weighted vector. The issue is segmented into different terms (in this paper, a word means term) in which each element in the vector of the issue is the weight of a term, and the value stands for the importance of the term for the issue. Term frequency-inverse document frequency (*tf-idf*) is used to calculate weight. Tf-idf is based on two assumptions: First, the frequency of the appearance of a given word implies its importance to an issue. Second, the frequency of the appearance of a word in several issues causes it to become less useful to distinguish among these issues.

IV. METHODS

A. Text-based Classification

Facing the huge changes described in Section II-B, determining whether the regular pattern of free text found in research [4] still works, and whether the performance of the text-based classification model in dealing with large-scale projects is efficient, are required. Many text-based classifications are used to classify issues in different studies [4]–[6]. In this paper, various types of widely used text-based classifications, such as *Naive Bayes*, *Logistic Regression*, were selected to know which classifier performs best. Table II shows the selected text-based classifications and parameter settings, which are determined by the best performance of numerous tests.

TABLE II
TEXT-BASED CLASSIFICATIONS AND PARAMETERS SETTING

Classifier	API	Parameters Setting
SVM	SVC	kernel='linear'
NB	MultinomialNB	class_prior='None'
LR	LogisticRegression	penalty='l2'
RF	RandomForestClassifier	n_estimators=100, n_jobs=-1

A well-labeled dataset, which can be used to train the classification model is constructed by dataset labeling process and preprocessing. Table II shows the four classifiers built for each project. We use APIs of package *sklearn* to implement the methods. A ten-fold cross-validation was applied to separate dataset samples into training and testing sets to evaluate the classification model. The ten-fold cross-validation has a minimal effect on the sample characteristics and can investigate the stability of item loading on multiple factors.

B. Regression Analysis of Classification Performance

The classification performances of different projects are always various. This part aims to determine the factors that influenced the performance of text-based classification. Manual analysis was used to detect factors, and we discovered several misclassified issues containing both bug- and nonbug-prone parts. For example, contributors may discover some unreasonable designs or problems in the project when they submit an issue about feature request (*e.g.*, EX in Section IV-C). The issue EX is nonbug prone, but the part of problem description is bug prone, which may confuse the classification. Such issues are termed as ***confused issues***.

Regression analysis techniques were used to verify our perception. In this paper, multiple linear mixed effect models were used to investigate the factors that affect classifier performance. In addition to coefficients, the effect size of each variable obtained from ANOVA analyses was reported. The model's fit can be evaluated by pseudo R-squared, i.e., the marginal ($R_m{}^2$) and conditional ($R_c{}^2$) coefficient, to determine generalized mixed-effect models [24]. As implemented in the MuMIn package of R [25], ($R_m{}^2$) is the proportion of variance explained by the fixed effects alone, and ($R_c{}^2$) is the proportion of variance explained by the fixed and random effects. All numeric variables were first log transformed (plus 0.5 if necessary) to stabilize variance and reduce heteroscedasticity [26]. The variance inflation factors (VIFs) for each predictor were computed to test for multicollinearity. If the VIFs of all the remaining factors are below 3, then multicollinearity is absent [26].

1) Outcome: The outcome measure is *average F-measure* (calculated as Equation 4) of the classification. Because of ten-fold cross-validation that we use, we can obtain $10 * n_{project}$ records for regression analysis, where $n_{project}$ is the number of projects.

2) Predictors: Project- and issue-level measures were computed in this process. Project-level measures are features of the status of the project, whereas issue-level measures are features extracted from textual summary of issues.

Project-level measures

Star and watch: The number of stars and watches of the project. This can reflect the popularity of the project in GitHub.

Contributors: The number of developers active in the project. The data are acquired in the homepage of the project in GitHub.

Project age: The project duration from the creation, in timestamps.

Commits: The total number of commits of the project.

Issues: The number of issues used in training set.

Issue-level measures

Confused issues: The total number of confused issues. Each sentence of the issue is predicted using the best model built in Section IV-A. if not all sentences of the issue are predicted to the same part, the issue will be considered as a confused issue.

Median of words: The median number of words for each instance in the training set. More words are likely to contain more information, which may help for classifier.

C. Tow-stage Classification

Omitting structured information in GitHub, as described in section II-B, results in that less information can be used to build a synthesized classification model. The firsthand features can be extracted, except for free text of issues, are relating to the historical information about issue contributors, *e.g.*, contributor's identity (core or external developer) and historical developing activities. Thus, we propose a two-stage

classification approach to combine textual summary information and developer information, which could be expected to improve the performance of classification. Each stage of our approach is explained in the following paragraphs, and the overview of our approach is shown in Figure 5.

1) Stage 1 - Textual Summary Classification: In this stage, the main task is to extract the information in the free text. Similar to the process of Section IV-A, two main textual information sources, title and description, were used. The classification model was trained from the textual information of training set, and the probability output of the model was applied to predict the testing set. After that, the title and description of the issue were divided into sentences, and the classification model we build before was used to predict each sentence. The sentence prediction results explain the changes in semantic when contributors report an issue. Free text can be examined further by analyzing the semantic perplexity information of the sentence and the regular pattern in submitting issues. The example will explain semantic perplexity.

EX: "*Currently, auto-archiving cannot be used if Piwik's authentication is configured to use the CAS plugin. I ran into this problem with authentication when running archive.php with CAS plugin enabled on my site... Add a feature to auto-archiving, so that it can succeeds when Piwik uses CAS for authentication instead of the default Login module.*"

Based on the EX, the first sentence describes the problem encountered by the contributor. For the classification model, this sentence is more likely to be predicted as bug-prone. However, a new feature is proposed in the last sentence, which is more likely to be predicted as non-bug-prone. For the classification model, issues similar to EX are difficult to classify because of the perplexity of the text. This situation can be addressed by extracting the features and dividing the issue into sentences. In Stage 1, the following features from the textual summary were extracted:

Probability: The probability that the issue report is predicted as bug-prone. The probability output of the classification model is used to obtain this feature.

SentenceCount: The total number of sentences in the issue report, including title and description.

MostBugProb: The maximum probability of all sentences that are predicted as bug-prone.

MostNonbugProb: The maximum probability of all sentences that are predicted as nonbug-prone.

Location: The sequence number of the most non-bug sentence. This feature was used to show where the nonbug sentence appears in the issue report.

BugCount: The number of sentences that are predicted as bug-prone.

NonbugCount: The number of sentences that are predicted as non-bug-prone.

ChangeCount: The number of semantic changes. For sentences sequence, every time the prone of sentences change from bug-prone to nonbug-prone or vice versa, the semantic changes will add one.

Perplexity: The perplexity of the issue. For sentence sequences of the issue, a series of probabilities that are predicted as bug-prone were collected and their perplexity were calculated using Equation 2, which is borrowed from the perplexity of natural language processing (NLP).

$$Perplexity = \frac{1}{SentenceCount} \sum \log(p_{i+1} - p_i) \quad (2)$$

Where p_i is the probability of the ith sentence.

These features need to be carefully extracted because of using the ten-fold cross-validation. All these features are assumed to be produced by a training set, which means that the training set was used to build a prediction model, and use this model to extract features of each instance in the training set. This approach allows the extraction of features without using the labeled information of a testing set, which may introduce some additional information and produce less-scientific results.

2) Stage 2 - Combining Free Text and Developer Information Classification: In Stage 1, the probability of bug-prone and perplexity information of sentences for each issue were obtained from free text. These features will be part of the input of Stage 2.

The experience of developers may influence the categories of issues. For example, skilled developers are likely to report a bug-prone issue and provide issue reports that meet the specification of the core team. Thus, in Stage 2, some structured features about contributors who submit issue reports were provided. These features contain identity of contributors in the project, historical developing activities, and social influence. The detailed information are as follows:

IsCoreTeam: This feature shows whether the contributor who reported the issue is a core team member in the project. If the contributor is in the core team, this feature is set to 1, otherwise, 0.

IssueCountInProject: The number of issues the contributor reports before in the project.

IssueCountInGitHub: The number of issues the contributor reports to GitHub.

CommentCountInProject: The number of comments that the contributor commits in the project.

CommentCountInGitHub: The number of comments the contributor commits in GitHub.

FollowerCount: The number of followers that the contributor acquires. This feature can reveal the social influence of the contributor in GitHub.

RegisterTime: This feature shows the duration that the contributor has registered. The longer a contributor has registered, the more familiar he is with the principles of GitHub.

Logistic regression was used as our prediction model in Stage 2. Logistic regression needs to be careful in partitioning datasets into training and testing sets before a prediction model is built. The output of Stage 1 and input of Stage 2 are associated. The testing set of Stages 1 and 2 should be similar to ensure that the same training set will be used to build the model, and to avoid the introduction of extra information from the testing set.

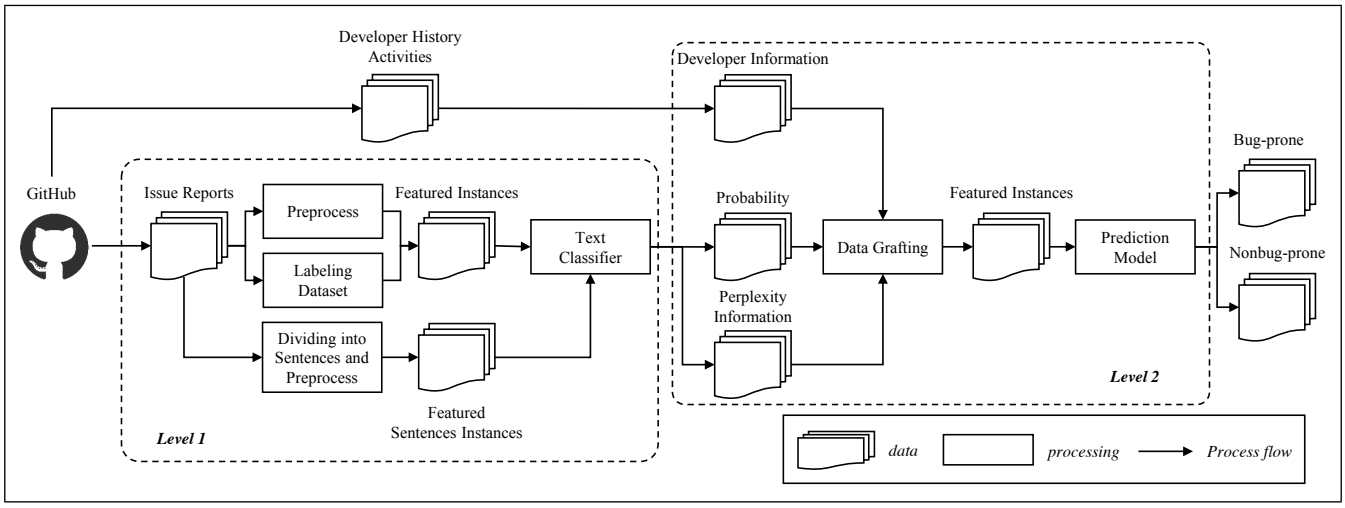

Fig. 5. Overview of two-stage classification framework

V. RESULTS AND DISCUSSION

A. Evaluation Metrics

Precision, *recall* and *F-measure* are widely used standard metrics in related work, such as issue assignment [27], bug prediction [8], [28] and reviewer recommendation [29], [30]. These metrics can measure the performance of models from different perspectives. For instance, *precision* is used to measure the exactness of the prediction, whereas *recall* evaluates the completeness.

F-measure denotes the balance and discrepancy between *precision* and *recall*, which can be interpreted as the weighted average of *precision* and *recall*:

$$F - measure = 2 * \frac{precision * recall}{precision + recall} \tag{3}$$

In [6], the weighted average value of *F-measure* for both categories is used to evaluate the classification model. This metric considers the performance of both categories and provides an overall performance of the classification model. Thus, in this paper, metrics similar to [6], which are defined in Equation 4, were selected, in which the *average F-measure* as f_{avg}, *F-measure* of bug (nonbug) as f_{bug} (f_{nonbug}), and number of bug (nonbug) as n_{bug}(n_{nonbug}).

$$f_{avg} = \frac{n_{bug} * f_{bug} + n_{nonbug} * f_{nonbug}}{n_{bug} + n_{nonbug}} \tag{4}$$

B. RQ1:Performance of Text-based Classification

For issue classification, four different machine learning classifiers ("*Naive Bayes*", "*Logistic Regression*", "*Random Tree*", "*Support Vector Machine*") were used on our dataset, and four classification models were built for each project to discuss which text-based classifier performs best. A baseline method was set, a grep-based method, which uses a simple grep with keywords like "bug","defect", or "fix" to determine whether an issue is bug. Boxplot was used to exhibit the result of each classifier and acquire the overall performance of all projects for each classifier. The *average F-measure* (*i.e.*, f_{avg}) was used to evaluate these classifiers and calculate the average value of ten-fold results as performance. Figure 6 shows the f_{avg} of baseline method and four different classifiers, where the y-axis is f_{avg}.

Figure 6 shows that all four classifier approaches outperform the baseline method. *Logistic Regression* and *Random Tree* reach a close performance, which are better than that of *Naive Bayes* but slightly worse than *SVM*.

Statistical analysis was used to verify our conclusions about the difference between text-based classifiers and the baseline method. Traditionally, the comparison of multiple groups follows a two-step approach: first, a global null hypothesis is tested, and then multiple comparisons are used to test the sub-hypotheses pertaining to each pair of groups. However, the global test null hypothesis may be rejected, whereas none of the sub-hypotheses are rejected, or vice versa [31]. Therefore, the one-step approach, multiple contrast test procedure $\widetilde{\mathbf{T}}$ [32], [33], is preferred in this study. The procedure $\widetilde{\mathbf{T}}$ by *nparcomp* package [34] in R was implemented to evaluate the F-Measure of all the approaches operating on 80 projects in our dataset. The *Tukey* (all-pairs) was set to contrast to compare all groups pairwise. For each pair of groups, the 95% confidence interval was analyzed to test whether the corresponding null sub-hypothesis can be rejected. If the lower boundary of the interval is greater than zero for groups A and B, then the metric value is higher in A than in B. Similarly, if the upper boundary of the interval is less than zero for groups A and B, then the metric value is lower in A than in B. Finally, if the lower boundary of the interval is less than zero and the upper boundary is greater than zero, then the data do not provide sufficient evidence to reject the null hypothesis.

Table III shows results of procedure $\widetilde{\mathbf{T}}$ (the last three rows are the results of Section V-D). All the *p-values* are less than 0.05 in the first four rows. Thus, a significant difference among the four text-based classifications and base line method is observed, which implies that text-based classifications are

TABLE III
COMPARISONS OF DIFFERENT TEXT-BASED CLASSIFIERS

Group A vs. Group B	Estimator	Lower	Upper	Statistic	p-value
NB vs. Base Line	0.028	0.010	0.074	-9.61727834	0.000000e+00
LR vs. Base Line	0.005	0.001	0.025	-8.85487385	0.000000e+00
RF vs. Base Line	0.001	0.000	0.016	-6.53029671	3.446741e-10
SVM vs. Base Line	0.001	0.001	0.002	-43.36487241	0.000000e+00
SVM vs. NB	0.127	0.067	0.226	-7.62243934	0.1.173506e-13
SVM vs. LR	0.310	0.207	0.437	-4.04251177	5.212218e-04
SVM vs. RF	0.296	0.195	0.421	-4.36384024	1.115098e-04
Combined method vs. SVM	0.348	0.246	0.466	-3.277566	0.005725885
Combined method vs. developer information	0.398	0.290	0.517	-2.207413	0.012257345
Combined method vs. perplexity information	0.448	0.336	0.567	-1.113729	0.048434745

useful for the issue classification in ITS of GitHub. Similarly, all the *p-values* are less than 0.05 in the next four rows and the lower and upper boundaries are greater than zero, which means that the performance of *SVM* is significantly better than the other three classifications.

Result 1: *In the context of GitHub's ITS, text-based classification approaches can achieve 69.7% to 98.9% of average F-measure (calculated as Equation 4) on our large-scale dataset, and the SVM classifier is the most effective approach compared to other typical classifiers.*

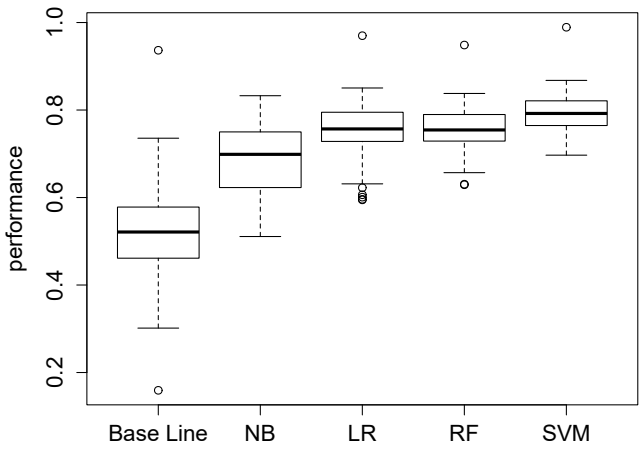

Fig. 6. f_{avg} of different ML methods

C. RQ2:Regression Analysis

Table IV shows the result of regression analysis. The model achieves an amazing fit ($R_c{}^2 = 92.5\%$). In our model, all the remaining factors are well below three, thereby indicating the absence of multicollinearity [26]. Moreover, no interaction among the variables in the models is observed, making the interpretation of our results easy and maintaining the cleanliness of the models. For project-level measures, the number of issues ($log(issue\_num)$) is highly significant, which means that the number of issues in train set is far from sufficient. Based on current data set, the more training sets used, the higher is the *average F-measure* that the classification achieves. Moreover, no statistical significance is observed in other project-level measures with regard to the influencing the performance of the classification model. For issue-level measures, the number of confused issues ($log(confuse\_count + 0.5)$) contained in the dataset is highly significant. When the dataset contains many confused issues, too many instances locate closely at the hyperplane of the classification model, which complicates the construction of an effective model. The median number of words ($log(med\_word\_count)$) in issues is insignificant, thereby suggesting that providing many textual summaries of the issues does not help in distinguishing between bug-prone or non-bug-prone, and some key words may be enough to build an effective classification model.

TABLE IV
REGRESSION RESULT OF FIXED EFFECT

	Coeffs	Sum Sq.
(intercept)	-3.37543*	
$log(star + watch)$	0.06316	0.197
$log(issue\_num)$	1.90440***	30.484***
$log(contributors)$	-0.03135	0.022
$log(age + 0.5)$	-0.22421*	0.288
$log(commits)$	-0.34256	0.842*
$log(confuse\_count + 0.5)$	-1.83346***	134.623***
$log(med\_word\_count)$	0.12505	0.067
marginal R-squared	0.6798150	
conditional R-squared	0.9251896	
signif.: $p < 0.001$ '***', $p < 0.01$ '**', $p < 0.05$ '*'		

Result 2: *Increasing the size of the training set can effectively improve the performance of the classification model. Furthermore, too many confused issues in the training set will seriously affect its performance.*

D. RQ3:2-stage Classification

An experiment based on the two-stage classifier was conducted to validate our approach. In the first stage, SVM was used as text classifier based on the conclusion of RQ1. In the second stage, we selected Logistic Regression as our prediction model, which performed better than other classifier in table II. As projects that achieve a high f_{avg} (*i.e.*, average

F-measure) contain few confused issues, our approach has a slight effect on these projects. Thus, to explore the performance of our approach for different projects, the project selection has two cases. In the first case, projects whose f_{avg} is less than the *first quartile* (0.7521) are selected. In the second case, projects whose f_{avg} is less than the *median* (0.7935) are selected. In this paper, ***SVM*** is selected as the first method, whose f_{avg} is the best among the four different text-based classifiers. Two other approaches are used to explore the effect of ***developer information*** and ***perplexity information*** in our two-stage approach. The developer information method only extracts probability of being bug-prone from free text in the first stage, because omitting the structured information is serious in GitHub; thus, the historical activities of the reporter in Stage 2 were selected to build as a classifier similar to work [6] and are described in Section IV-C2. The perplexity information method, which is used to compare the effect of perplexity information and developer information, extracts perplexity in the first stage, and does not use structured developer information in the second stage. The ***combined method*** is our two-stage approach, which makes use of both perplexity information and developer information. Figure 7 shows the comparison results of f_{avg}.

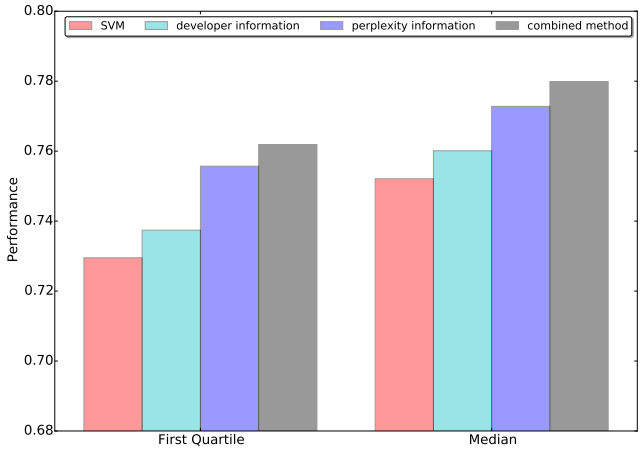

Fig. 7. f_{avg} of different methods

Figure 7 shows that the combined method outperforms all other methods for 80 projects. For procedure $\widetilde{\mathbf{T}}$ (last rows in Table III), all the *p-values* of combined method versus SVM, developer information method and perplexity information method are less than 0.05, and the lower and upper boundaries are greater than zero, which means that combined method is more significant than other approaches.

Result 3: *The two-stage classification approach can achieve a statistically significant improvement compared to traditional text-based classification by integrating our novel perplexity features.*

Although the value of the absolute increase is not impressive, the extracted features (semantic perplexity information) are generally effective in improving the performance of the classifier model. The result of dividing sentences is not ideal. Issue reports in these projects are contrasted sharply, combining free text with codes, hyper-link, and stack track, thereby complicating the finetuning for every project. The preprocessing is not perfect, which limits the promotion of some projects. Even facing this challenge, our approach still achieves a stable improvement. We believe that when applied in practice, a highly individualized data preprocessing approach can be a great help in extracting features that are in agreement with our approach.

VI. Threats to Validity

Our study had two main threats. The fist threat concerned the study design about category extraction on our data set. We utilized tags used most in GitHub to distinguish the category of issues. We trained model on issues with these tags, so that we could clearly know the category of samples in training set. We ignored the issues without these tags, which might introduce bias to dataset. However, many researches [4], [6] selected labeled issues as training set. What's more, unlabeled issues was only a small part of the data set, which was 9.8% in our study.

The second threat is the number of projects we used. We filtered projects by the number of labeled issues and the rate of bug-prone issues in projects. Our findings are based on 80 projects in GitHub. Compared to other studies, although we used a large-scale dataset, it is still a very small part of projects in GitHub. For filtered projects, our method need an extra adjustment and further evaluation.

VII. Conclusions

Management tasks are always cumbersome and repetitive. Using automatic methods to assist in solving these repetitive but necessary tasks is crucial. In this paper, automatic classification techniques for issue reports were examined.

Four different text-based classification approaches in a large-scale dataset were used to determine which approach performs best. Regression analysis by multiple linear mixed effects models was used to investigate key factors that limit the performance of the classifier, and discovered that semantic perplexity is a crucial factor that affects the performance of classification. A two-stage classifier framework was established based on the regression analysis. Some experiments were conducted and the results show the following:

1) In our large scale study, text-based classification approaches work in the context of GitHubs ITS, and Support Vector Machines (SVM) achieves the best performance among 4 different text-based classifiers.

2) Increasing the size of training set can effectively improve the performance of the classification model. Besides, too many confused issues (defined in Section IV-B) are harm for building an effective text-based classifier.

3) Our 2-stage classifier framework can extract semantic perplexity information from free text, which is benefit for classifier. The quantitative evaluations show that the classification performance can achieve a significant improvement.

In our future work, we plan to explore the relationship between the performance of classifier and topics of the issue report. We believe that the distribution of categories is associate with topics of issue reports. Otherwise, a more outperforming preprocessing is our sustained progressing task.

ACKNOWLEDGMENT

This research is supported by National Science Foundation of China (Grant No.61432020, 61472430, 61502512 and 61303064) and National Key R&D Program of China (2016YFB1000805).

REFERENCES

[1] G. Gousios, A. Zaidman, M.-A. Storey, and A. van Deursen, "Work practices and challenges in pull-based development: The integrators perspective," in *Proceedings of the 37th International Conference on Software Engineering*, vol. 1, 2015, pp. 358–368.

[2] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*. IEEE, 2008, pp. 52–61.

[3] W. Huang, T. Lu, H. Zhu, G. Li, and N. Gu, "Effectiveness of conflict management strategies in peer review process of online collaboration projects," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. ACM, 2016, pp. 717–728.

[4] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*. ACM, 2008, p. 23.

[5] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: how misclassification impacts bug prediction," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE, 2013, pp. 392–401.

[6] Y. Zhou, Y. Tong, R. Gu, and H. Gall, "Combining text mining and data mining for bug report classification," in *2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2014, pp. 311–320.

[7] K. Herzig, S. Just, A. Rau, and A. Zeller, "Predicting defects using change genealogies," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*. IEEE, 2013, pp. 118–127.

[8] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 2010, pp. 31–41.

[9] H. Hata, O. Mizuno, and T. Kikuno, "Bug prediction based on fine-grained module histories," in *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 2012, pp. 200–210.

[10] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in github," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 805–816.

[11] T. F. Bissyandé, D. Lo, L. Jiang, L. Reveillere, J. Klein, and Y. Le Traon, "Got issues? who cares about it? a large scale investigation of issue trackers from github," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*. IEEE, 2013, pp. 188–197.

[12] T. Zimmermann, R. Premraj, J. Sillito, and S. Breu, "Improving bug tracking systems." in *ICSE Companion*. Citeseer, 2009, pp. 247–250.

[13] J. Spolsky, "Painless bug tracking," 2010.

[14] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 361–370.

[15] O. Baysal, M. W. Godfrey, and R. Cohen, "A bug you like: A framework for automated assignment of bugs," in *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*. IEEE, 2009, pp. 297–298.

[16] 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*. ACM, 2008, pp. 461–470.

[17] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 2010, pp. 45–54.

[18] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Proceedings of the Fourth International Workshop on Mining Software Repositories*. IEEE Computer Society, 2007, p. 1.

[19] T. Merten, M. Falis, P. Hübner, T. Quirchmayr, S. Bürsner, and B. Paech, "Software feature request detection in issue tracking systems," in *Requirements Engineering Conference (RE), 2016 IEEE 24th International*. IEEE, 2016, pp. 166–175.

[20] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, "Wait for it: Determinants of pull request evaluation latency on github," in *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*. IEEE, 2015, pp. 367–371.

[21] G. Gousios, "The ghtorent dataset and tool suite," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 233–236.

[22] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman, "Lean ghtorrent: Github data on demand," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 384–387.

[23] W. B. Frakes and R. Baeza-Yates, "Information retrieval: data structures and algorithms," 1992.

[24] J. Tsay, L. Dabbish, and J. Herbsleb, "Let's talk about it: evaluating contributions through discussion in github," in *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*. ACM, 2014, pp. 144–154.

[25] J. J. Jiang, G. Klein, H.-G. Hwang, J. Huang, and S.-Y. Hung, "An exploration of the relationship between software development process maturity and project performance," *Information & Management*, vol. 41, no. 3, pp. 279–288, 2004.

[26] M. Gharehyazie, D. Posnett, B. Vasilescu, and V. Filkov, "Developer initiation and social interactions in oss: A case study of the apache software foundation," *Empirical Software Engineering*, vol. 19, no. Part C, pp. 342–354, 2014.

[27] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer, "Categorizing bugs with social networks: a case study on four open source software communities," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 1032–1041.

[28] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 529–540.

[29] J. B. Lee, A. Ihara, A. Monden, and K.-i. Matsumoto, "Patch reviewer recommendation in oss projects," in *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, vol. 2. IEEE, 2013, pp. 1–6.

[30] G. Jeong, S. Kim, T. Zimmermann, and K. Yi, "Improving code review by predicting reviewers and acceptance of patches," *Research on Software Analysis for Error-free Computing Center Tech-Memo (ROSAEC MEMO 2009-006)*, pp. 1–18, 2009.

[31] K. R. Gabriel, "Simultaneous test procedures–some theory of multiple comparisons," *The Annals of Mathematical Statistics*, pp. 224–250, 1969.

[32] F. Konietschke, L. A. Hothorn, E. Brunner *et al.*, "Rank-based multiple test procedures and simultaneous confidence intervals," *Electronic Journal of Statistics*, vol. 6, pp. 738–759, 2012.

[33] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment?" *Information and Software Technology*, vol. 74, pp. 204–218, 2016.

[34] J. R. Fraenkel, N. E. Wallen, and H. H. Hyun, *How to design and evaluate research in education*. McGraw-Hill New York, 1993, vol. 7.