2015

Automatic, high accuracy prediction of reopened bugs

Xin Xia

David LO
Singapore Management University, davidlo@smu.edu.sg

Emad Shihab

Xinyu Wang

Bo Zhou

DOI: https://doi.org/10.1007/s10515-014-0162-2

Autom Softw Eng
DOI 10.1007/s10515-014-0162-2

Automatic, high accuracy prediction of reopened bugs

Xin Xia · David Lo · Emad Shihab · Xinyu Wang · Bo Zhou

Received: 3 October 2013 / Accepted: 25 July 2014

Abstract Bug fixing is one of the most time-consuming and costly activities of the software development life cycle. In general, bugs are reported in a bug tracking system, validated by a triage team, assigned for someone to fix, and finally verified and closed. However, in some cases bugs have to be reopened. Reopened bugs increase software maintenance cost, cause rework for already busy developers and in some cases even delay the future delivery of a software release. Therefore, a few recent studies focused on studying reopened bugs. However, these prior studies did not achieve high performance (in terms of precision and recall), required manual intervention, and used very simplistic techniques when dealing with this textual data, which leads us to believe that further improvements are possible. In this paper, we propose *ReopenPredictor*, which is an automatic, high accuracy predictor of reopened bugs. *ReopenPredictor* uses a number of features, including textual features, to achieve high accuracy prediction of reopened bugs. As part of *ReopenPredictor*, we propose two algorithms that are used to automatically estimate various thresholds to maximize the prediction performance.

X. Xia · X. Wang (✉) · B. Zhou
College of Computer Science and Technology, Zhejiang University, Hangzhou, Zhejiang, China
e-mail: wangxinyu@zju.edu.cn

X. Xia
e-mail: xxkidd@zju.edu.cn

B. Zhou
e-mail: bzhou@zju.edu.cn

D. Lo
School of Information Systems, Singapore Management University, Singapore, Singapore
e-mail: davidlo@smu.edu.sg

E. Shihab
Department of Computer Science and Software Engineering, Concordia University, Montreal, QC, Canada
e-mail: eshihab@cse.concordia.ca

Published online: 18 September 2014

To examine the benefits of *ReopenPredictor*, we perform experiments on three large open source projects—namely Eclipse, Apache HTTP and OpenOffice. Our results show that *ReopenPredictor* outperforms prior work, achieving a reopened F-measure of 0.744, 0.770, and 0.860 for Eclipse, Apache HTTP and OpenOffice, respectively. These results correspond to an improvement in the reopened F-measure of the method proposed in the prior work by Shihab et al. by 33.33, 12.57 and 3.12 % for Eclipse, Apache HTTP and OpenOffice, respectively.

Keywords Reopened bugs · Imbalanced feature selection · Imbalanced learning

1 Introduction

Bug fixing is one of the main activities in the software development and maintenance process. The vast majority of open source and commercial software projects use bug tracking systems, such as Bugzilla, to manage their software bugs. Since effective management of bugs is a very important problem, numerous research studies have proposed automated techniques to manage bug assignments, such as bug triaging (Anvik et al. 2006), the detection of duplicate bug reports (Sun et al. 2011), etc.

However, in some cases, a bug has to be reopened. Bugs can be reopened for various reasons (Zimmermann et al. 2012); for example, if a tester did not provide sufficient information in the bug report, if the developer misunderstood the root cause of the bug, if the bug reappeared in the current version of system although it was fixed in the previous version (i.e., a regression bug), and so on. Reopened bugs are undesirable since they take longer to resolve (almost twice as long as not reopened bugs), they consume valuable time from the already-busy developers and lead end users to lose trust in the quality of the software product (Shihab et al. 2010). Such negative impact of reopened bugs has been confirmed in both open source (e.g., Shihab et al. 2010, 2012; Xia et al. 2013) and commercial projects (Zimmermann et al. 2012).

Recently, a few studies investigated reopened bug reports. The most related studies are Shihab et al. (2010, 2012), Xia et al. (2013), and Zimmermann et al. (2012). Shihab et al. (2010, 2012) propose the problem of predicting reopened bug reports. Xia et al. (2013) investigate the effectiveness of different supervised learning algorithms for reopened bug prediction. Zimmermann et al. (2012) comprehensively investigate factors that cause bug reports to be reopened in Microsoft. Although these prior studies focused on reopened bugs, they were mostly exploratory in nature. First, we note that their precision and recall need to be improved in order to make them applicable in a real-life scenario. Second, most of these studies required manual intervention and tuning in order to determine certain thresholds that make their approaches work.

A major challenge that causes low precision and recall when reopened bugs are considered is the fact that only a small proportion of bug reports are actually reopened. There is an unequal distribution between reopened and non-reopened bug reports. Only 16.1, 6.46, and 26.31 % of the bug reports in the whole bug report repository of Eclipse, Apache HTTP, and OpenOffice projects respectively are reopened (for more details, see Table 2). We refer to this phenomenon as the *class imbalance phenomenon* (He and Garcia 2009). Due to the class imbalance phenomenon, predicting reopened bugs with high accuracy is a difficult task.

Therefore, in this paper we propose *ReopenPredictor*, which improves the performance of existing techniques. Past approaches (e.g., Shihab et al. 2010, 2012; Xia et al. 2013) only consider a limited set of features, ignore the class imbalanced phenomenon, and uses only one classifier. *ReopenPredictor* achieves better performance over previous approaches by the following means. First, we investigate various kinds of features that we could extract from the dataset; these include: features from raw textual information contained in the summary and description fields of bug reports (description features), features from raw textual information in the comments of bug reports (comment features), and features proposed by Shihab et al. (meta features). Second, to overcome the issue related to the high dimensionality of textual data and the class imbalance phenomenon, we propose a feature selection method which is based on the correlation coefficient. Third, *ReopenPredictor* is a composite classification-based framework, which automatically combines and assigns weights to multiple classifiers (i.e., a description classifier, a comment classifier, and a meta classifier), to achieve a high performance. A developer can use our tool at the end of a bug fixing process. If our tool predicts that a bug report would be reopened, the developer can then put more effort to mitigate the reopening of the bug. For example, a developer can perform additional code reviews or to add more test cases to ensure that the bug has been properly fixed.

In this work we integrate a relatively new software engineering (SE) problem, namely "predicting reopened bugs" with a specialized artificial intelligence (AI) solution which adapts a machine learning (a subfield of AI) algorithm to perform well on the SE problem. The research problem motivates the need for a new adaptation of the machine learning algorithm, and the adapted algorithm improves the performance of the automated software engineering task. Thus, we synergize SE and AI together.

The main contributions of this paper are as follows:

(1) We propose a novel composite method, *ReopenPredictor*, which blends three different classifiers trained using three different sets of features, to achieve better performance for reopened bug prediction. The experiment results show that our proposed *ReopenPredictor* achieves a much better performance than the method used in (Shihab et al. 2010, 2012).
(2) We investigate the usefulness of the bug report text information for reopened bug prediction. In doing so, we propose an imbalanced feature selection method that can automatically select the most substantial textual features.
(3) We contribute to the large body of empirical work by replicating the work on reopened bug prediction (Shihab et al. 2010, 2012), strengthening our confidence in the findings of empirical studies.

The remainder of the paper is organized as follows. In Sect. 2, we present the motivation of reopened bug prediction. In Sect. 3, we present an overall framework of our *ReopenPredictor*. In Sect. 4, we propose our feature selection method which chooses important textual features to predict whether a bug report will be reopened or not. In Sect. 5, we propose *RPComposer*, a composite method, which uses all the 3 types of features, to achieve better performance. In Sect. 6, we report the results of our experiment which compares our proposed approach with the algorithms proposed

by Shihab et al. In Sect. 7, we present related studies. In Sect. 8, we conclude and mention future work.

2 Motivation

In general, a typical bug goes throughout the following four steps (we take Bugzilla as an example):

(1) A tester or an end user detected a bug in the software, and published a bug report to describe the bug in Bugzilla. The current bug status is "new";
(2) A bug triager assigned this bug to the most appropriate developer. The current bug status is "assigned";
(3) The developer read the bug report, and verified whether it is a real bug, and tried to fix it. The current bug status is "verified";
(4) Finally, after the developer had fixed the bug or verified that it is not a bug, the bug triager closes this bug. The current bug status is "closed";

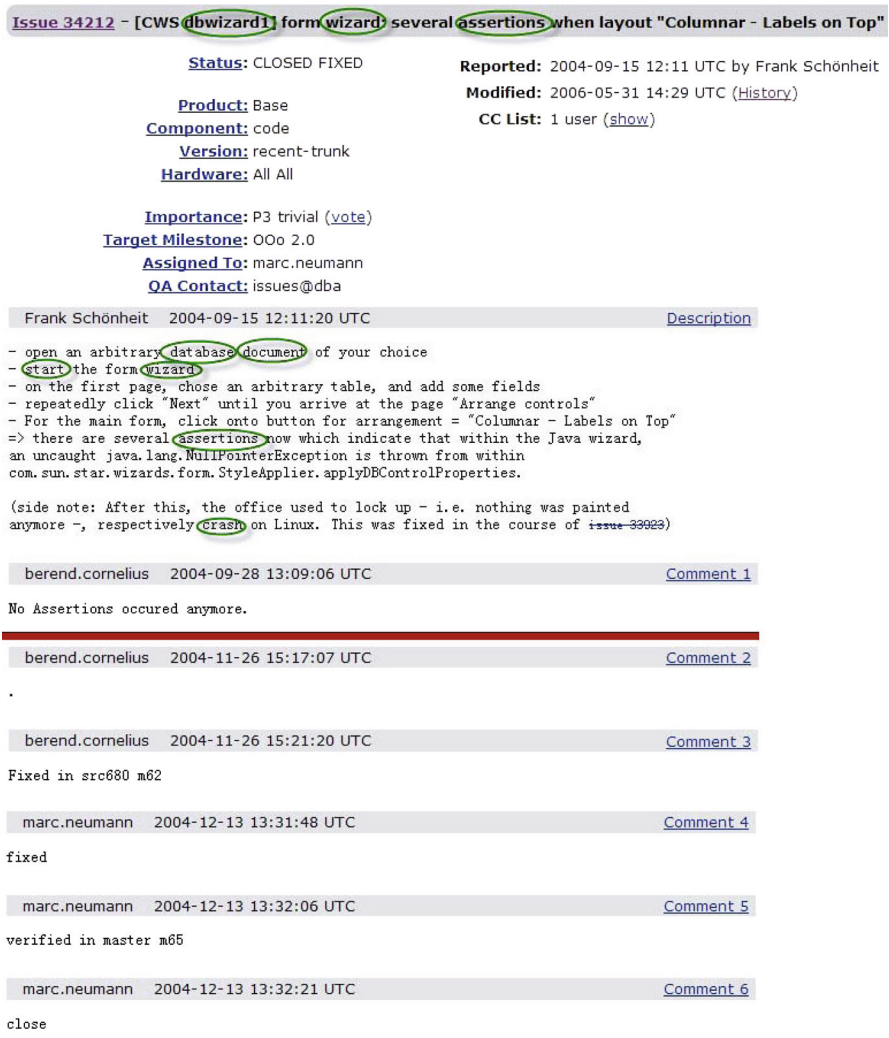
However, in certain cases a bug is reopened. For example, Fig. 1 presents a reopened bug report of OpenOffice project with BugID 41356. We list the summary and description fields of the bug report, the list of comments before the bug report is reopened, as well as other fields in the bug report such as product, component, report time, etc.

Reopened bugs can reduce the reliability of a software system. A past study by Shihab et al. (2012) has also shown that reopened bugs take longer to be fixed. The example in Fig. 1 shows that the bug is reopened around two months after it was closed. In these two months, the software is still buggy. The bug fixing process is also delayed by more than two months. After the two months, the bug fixer needs to restart the bug fixing effort which involves trying to understand what the bug is about, what the code is supposed to do, and so on. With *ReopenPredictor*, a developer can use our tool at the end of a bug fixing process (i.e., at 2004-09-28). If our tool predicts that a bug report would be reopened, the developer can then put more effort to perform code review or to add more test cases to ensure that the bug has been properly fixed. Thus, the bug would not be reopened, and it will make the system more reliable and reduce unnecessary maintenance cost.

It is difficult to decide which words are more important in predicting that a bug report would be reopened or not. With our imbalanced feature selection method we can consider other bug reports that are reopened and those that are not, and use them to identify important words, e.g., "document", "start", "wizard", "crash", "database", and "dbwizard".[1]

The aforementioned scenario is just one example, however, we believe that this example gives us a hint that the textual comments in the bug report contain important information. Therefore, in contrast to past research on reopened bugs, a large portion of our work focuses on how to optimally leverage textual information in order to improve the performance of the reopened bug prediction. As it turns out, leveraging textual information is not an easy task due to the class imbalance phenomenon, which we discuss in detail in Sect. 4.2.

[1] For more details, please refer to Sect. 4.


Issue 34212 - [CWS dbwizard1] form wizard: several assertions when layout "Columnar - Labels on Top"

Status: CLOSED FIXED
Reported: 2004-09-15 12:11 UTC by Frank Schönheit
Modified: 2006-05-31 14:29 UTC (History)
CC List: 1 user (show)

Product: Base
Component: code
Version: recent-trunk
Hardware: All All

Importance: P3 trivial (vote)
Target Milestone: OOo 2.0
Assigned To: marc.neumann
QA Contact: issues@dba

Frank Schönheit 2004-09-15 12:11:20 UTC Description

- open an arbitrary database document of your choice
- start the form wizard
- on the first page, chose an arbitrary table, and add some fields
- repeatedly click "Next" until you arrive at the page "Arrange controls"
- For the main form, click onto button for arrangement = "Columnar - Labels on Top"
=> there are several assertions now which indicate that within the Java wizard,
an uncaught java.lang.NullPointerException is thrown from within
com.sun.star.wizards.form.StyleApplier.applyDBControlProperties.

(side note: After this, the office used to lock up - i.e. nothing was painted
anymore -, respectively crash on Linux. This was fixed in the course of ~~issue 33923~~)

berend.cornelius 2004-09-28 13:09:06 UTC Comment 1

No Assertions occured anymore.

berend.cornelius 2004-11-26 15:17:07 UTC Comment 2

.

berend.cornelius 2004-11-26 15:21:20 UTC Comment 3

Fixed in src680 m62

marc.neumann 2004-12-13 13:31:48 UTC Comment 4

fixed

marc.neumann 2004-12-13 13:32:06 UTC Comment 5

verified in master m65

marc.neumann 2004-12-13 13:32:21 UTC Comment 6

close

Fig. 1 Reopened bug report of OpenOffice Project with BugID 34212. The comments below the *dark line* are the comments after the bug report is reopened

3 Overall framework

Figure 2 presents the overall framework of *ReopenPredictor*. The entire framework contains two phases: a model building phase and a prediction phase. In the model building phase, our goal is to build a model from historical bug reports that can discriminate reopened from non-reopened bug reports. In the prediction phase, this model is used to predict if an unknown bug report would be reopened or not.

Our framework first collects various fields from a set of training bug reports with a known status (Steps 1, 2 and 3). We collect the description and summary text fields (Step 1), the comments text fields (Step 2), and other fields (such as previous status,

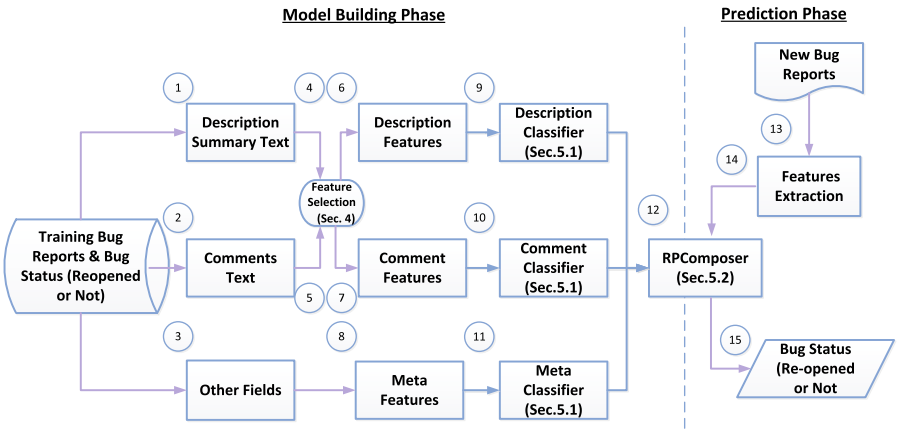


Fig. 2 Overall framework for reopened bug prediction (*ReopenPredictor*)

priority, component, product, report time, etc) (Step 3). It is important to note that we only consider comments before a bug report was reopened. For the description and summary text, and comments text, we tokenize them, remove the stop-words, stem them, and represent them in the form of "bags of words" (Steps 4 and 5). Each processed word becomes a feature. Then we use our feature selection method to generate substantial features from the description and summary text, and the comments text, respectively (Steps 6 and 7).[2] Moreover, for the other fields, we extract meta features (Step 8). These features are first proposed in (Shihab et al. 2010, 2012) and are briefly described in Table 1.

Next, our framework constructs the classifiers based on the three types of features (Steps 9, 10 and 11). A classifier is a machine learning model which assigns labels (in our case: reopened or not) to a data point (in our case: a bug report) based on its features.[3] We then blend or combine the 3 classifiers together to construct a *RPComposer* classifier (Step 12).[4]

After the *RPComposer* classifier is constructed, in the prediction phase they are then used to predict whether a bug report with an unknown label will be reopened or not. For each such bug report, we first extract the values of the features belonging to the three feature types from it (Step 13). We then input the feature values to the *RPComposer* classifier (Step 14) that outputs the prediction result, which is one of the following labels: reopened or non-reopened (Step 15).

4 Imbalanced text feature selection

We consider three sources of information for reopened bug prediction. Two of these are textual information that are represented as bags-of-words. Each processed word in

[2] For more details, please refer to Sect. 4.

[3] For more details of these three classifiers, please refer to Sect. 5.1.

[4] For more details of *RPComposer*, please refer to Sect. 5.2.

Table 1 Meta features extracted from bug reports as described in (Shihab et al. 2010, 2012)

Features	Type	Description
Time	Nominal	Hour the bug is closed
Weekday	Nominal	Day of the week the bug is closed
Month day	Numeric	Day of the month the bug is closed
Month	Numeric	Month the bug is closed
Day of Year	Numeric	Day of year when the bug is closed
Time days	Numeric	Time to solve the bug
Last status	Nominal	Last status of the bug
No. of source files	Nominal	Number of source code files related to the bug
Reporter Name	String	The bug reporter name
Fixer Name	String	The bug fixer name
Reporter Experience	Numeric	Number of bug reports reported by this reporter
Fixer Experience	Numeric	Number of bug reports fixed by this fixer
Component	Nominal	Component affected by the bug
Platform	Nominal	Platform affected by the bug
Severity	Numeric	Severity of the bug report
Priority	Numeric	Priority of the bug report
CC list Number	Numeric	Number of people in CC list
Description size	Numeric	Number of words in description text
Description text	Bayesian Scores	Description text content
No. of comments	Numeric	Number of comments
Comment size	Numeric	Number of words in comment text
Comment text	Bayesian Scores	Comment text content
Priority changed	Boolean	Whether the priority was changed
Severity changed	Boolean	Whether the severity was changed

the bags-of-words is a feature. Thus, we have a large number of features. In machine learning literature, a feature can be viewed as a dimension, and a data point (i.e., a bug report) can then be viewed as a point in this high-dimensional space. An overly high number of dimensions can cause what is known as the *curse-of-dimensionality* (Han and Kamber 2006). Additionally, we have the class imbalance problem since only a very small minority of bug reports are reopened. In this section, we address the above problems by proposing a new feature selection strategy that can be applied to the textual features of bug reports.

Our feature selection strategy builds on a well known feature selection metric referred to as the correlation coefficient (CC). We introduce the correlation coefficient and explain the rationale for using it in Sect. 4.1. We then propose our feature selection strategy that addresses the issues of high dimensionality and class imbalance in Sect. 4.2. Our proposed imbalanced feature selection method corresponds to the "Feature Selection" block in Fig. 2.

4.1 Correlation coefficient

In text classification, we have data points, features, and classes. In our settings, a bug report is a data point, a word is a feature, and there are two classes: reopened (+ve) and non-reopened bug reports (−ve). There are various feature selection metrics for text classification, such as information gain (IG), chi-square (CHI), correlation coefficient (CC), etc. (Zheng et al. 2004). Some of these metrics are one-sided, and some others are two-sided. A one-sided metric would assign a score to a feature based on how indicative a feature is to the target (+ve) class (in our case, reopened bug reports). With a one-sided metric, we could sort features from the ones which are most indicative to the target class to the least indicative features. A two-sided metric, on the other hand, would assign a score to a feature based on how indicative a feature is to either one of the classes (in our case, reopened bug reports and non-reopened bug reports). Thus we could not sort features based on how related they are with the target class. CC is a one-sided metrics, while IG and CHI are two-sided metrics.

In addition, as stated earlier, our dataset suffers from the class imbalance phenomenon: there are more data points with negative class label (i.e., non-reopened reports). Due to this, if we apply the two-sided metrics, we would get more features that are related to the −ve class than the +ve class. This would make the classifier learn from the resultant features to more likely predict −ve class for an unknown data point (i.e. bug report). Thus, this is another reason to use a one-sided metric such as CC.

Now we illustrate how the CC metric is calculated. Let us denote the bug report collection as $BR = \{(B_1, R_1), (B_2, R_2), ...(B_N, R_N)\}$, where B_i represents the i^{th} bug report and R_i is a label that represents whether this bug report is reopened (r) or not ($\bar{r}$) (i.e., $R_i \in \{r, \bar{r}\}$), and the words in BR as $T = \{t_1, t_2, ...t_m\}$. Then, the correlation coefficient of a term t to class r is defined as:

$$CC(t, r) = \frac{\sqrt{N}[P(t, r)P(\bar{t}, \bar{r}) - P(t, \bar{r})P(\bar{t}, r)]}{\sqrt{P(t)P(\bar{t})P(r)P(\bar{r})}} \quad (1)$$

In the above equation, $P(t, r)$, $P(t, \bar{r})$, $P(\bar{t}, r)$, $P(\bar{t}, \bar{r})$ represent the proportion of reopened bug reports that contain the word t, and the proportion of non-reopened bug reports that contain the word t, the proportion of reopened bug reports that do not contain the word t, and the proportion of non-reopened bug reports that do not contain the word t, respectively.

Note that CC of a term t can either be negative or positive. A negative value indicates that the word t is related to non-reopened bug reports. On the other hand, a positive value indicates that t is related to reopened bug reports.

4.2 Imbalanced feature selection

The CC computes a score for each word t. Words with large positive and negative values have opposite relationships with reopened bug reports. If a bug report has many words with large positive CC scores then there is a high likelihood that it will be reopened. On the other hand, if a bug report has many words with large negative CC scores, then

there is a high likelihood that it will not be reopened. The one-sided nature of CC makes it good for feature selection in our setting. However, the main question is: how should we select the positive and negative scored features for effective classification? One naive way is to select an equal number of features with the highest positive and negative scores. However, we propose a solution that is expected to perform better. Rather than fixing the proportion of features with the highest positive and negative scores, we tune this proportion using a training data. In this way, our solution considers the peculiarity of the domain (i.e., bug reports of a particular project) to decide for the best features to be selected.

Our proposed solution is to select l features where $m\,\%$ of them are features with the highest positive scores and the rest are features with the highest negative scores. We set the value of l to be high enough (i.e., 1,000, by default) so that a sufficient number of positive and negative terms are included. We reduce this number if there are relatively fewer words in a bug report collection. We divide the *training* dataset into two subsets: one is used to select l features, and another is used to evaluate the performance of the selected l features. The value of m is tuned by training and testing on the *training* dataset that we have in the model building phase. We vary the value of m using a grid search procedure (Bergstra and Bengio 2012): we start with a low value of m and increases it by a small amount, step-by-step.

Algorithm 1 presents our proposed method. First, we divide the bug collection BR into two subsets: BR_1 and BR_2 (Line 11). By default, we set the dividing coefficient as 90 %, i.e., we use 90 % of bug reports in BR_1 to select l words, and the remaining 10 % bug reports in BR_2 to evaluate the performance. Next, we compute the CC values for each word in BR_1, and choose l words with the largest positive CC scores (*candPos*) and l words with the largest negative CC scores (*candNeg*) (Lines 11, 12, and 13). Then, we select $m\,\%$ top words from *candPos* and $(100 - m)\,\%$ top words from *candNeg* (Lines 18–19). We build a classifier based on the selected features and perform testing on BR_2 to evaluate how good a value of m is. We record the reopened F-measure that is achieved by the classifier. Reopened F-measure is a common measure to evaluate how good a classifier is. It is the harmonic mean of precision and recall. Precision refers to the proportion of bug reports predicted as reopened that are correctly predicted. Recall refers to the proportion of reopened bug reports that are correctly identified. We repeat the process many times by increasing the value of m little-by-little, one step (i.e., 1) at a time. We record the value of m that gives us the highest F-measure.

5 RPComposer: a composite method

We collected three sets of features, i.e., meta features, description text features and comment text features. We build a classifier for each of these feature types. We then compose the three scores together in our proposed *ReopenPredictor* framework. In this section, first we define the three sets of scores outputted by the three classifiers. Next, we describe how we combine these scores together to construct the *RPComposer* classifier.

Algorithm 1 Imbalanced Feature Selection of Text Information in Reopened Bug Prediction

ImbalancedFeatureSelection(l,sp,BR)
Input:
l: Number of Words would be Selected
sp: Dividing Coefficient
BR: Training Bug Report Collection
Output: $positive, negative$
$positive$: Words with positive CC scores
$negative$: Words with negative CC scores
Method:
Divide BR into two subsets BR_1 and BR_2 according to sp;
Compute CC values for each word in BR_1;
candPos = l words with the largest positive CC scores from BR_1;
candNeg = l words with the largest negative CC scores from BR_1;
Best F-measure f;
Best Ratio m_{best};
Ratio Coefficient $m = 0$;
while $m \leq 100$ **do**
  Select m % top words from *candPos*, denoted as *tmpCandPos*;
  Select $(100 - m)$ % top words from *candNeg*, denoted as *tmpCandNeg*;
  Re-construct BR_1 and BR_2 with *tmpCandPos* and *tmpCandNeg*, denote as BR_1^{tmp} and BR_2^{tmp};
  Build a classifier from BR_1^{tmp} ;
  Evaluate the reopened F-measure f_{tmp} of the classifier using BR_2^{tmp};
  if $f_{tmp} > f$ **then**
    $f = f_{tmp}$;
    $m_{best} = m$;
  end if
  $m = m + 1$;
end while
Choose $positive$ from *candPos* with ratio m_{best} %;
Choose $negative$ from *candNeg* with ratio $(100 - m_{best})$ %;
Return $positive, negative$;

5.1 Feature scores

As illustrated in Fig. 2, our proposed framework has 3 different components which correspond to classifiers built using the three feature types. Let us refer to them as Cla_{Meta}, Cla_{Desc}, and $Cla_{Comment}$, which correspond to the "Meta Classifier", "Description Classifier", and "Comment Classifier" blocks in Fig. 2, respectively. Given an unknown bug report, Cla_{Meta}, Cla_{Desc}, and $Cla_{Comment}$ output the following meta scores, description scores, and comment scores, respectively:

Definition 1 (*Meta Scores*) Consider a training bug report collection BR, and its corresponding set of meta feature values $Meta$. We build a classifier Cla_{Meta} trained on $Meta$. For a new bug report br, we use Cla_{Meta} to get the likelihood that br will be reopened, and the likelihood that br will not be reopened. We refer to these likelihood scores as **meta scores**, and denote them as $Meta(br, r)$ and $Meta(br, \bar{r})$, respectively.

Definition 2 (*Description Scores*) Consider a training bug report collection BR, and its corresponding set of description feature values $Desc$. We build a classifier Cla_{Desc} trained on $Desc$. For a new bug report br, we use Cla_{Desc} to get the likelihood that

br will be reopened, and the likelihood that br will not be reopened. We refer to these likelihood scores as **description scores**, and denote them as $Desc(br, r)$ and $Desc(br, \bar{r})$, respectively.

Definition 3 (*Comment Scores*) Consider a training bug report collection BR, and its corresponding set of comment feature values $Comment$. We build a classifier $Cla_{Comment}$ trained on $Comment$. For a new bug report br, we use $Cla_{Comment}$ to get the likelihood that br will be reopened, and the likelihood that br will not be reopened. We refer to these likelihood scores as **comment scores**, and denote them as $Comment(br, r)$ and $Comment(br, \bar{r})$, respectively.

To compute the meta, description, and comment scores from a set of meta, description, and comment feature values respectively, a classification algorithm is used. The algorithm is trained on a training data of bug reports and it builds a classifier which is a machine learning model that can assign labels (in our case: reopened or not) to a new bug report with a certain likelihood. There are many classification algorithms proposed in the literature; most of them assign weights to the features and use the presence and absence of each of these features in a new bug report, along with the weights of the features to compute the likelihood of the new bug report to be assigned a particular label (i.e., reopened or not).

5.2 RPComposer

In this section, we propose *RPComposer*, a composite method which uses all of these three scores. A linear combination of meta scores, description scores, and comments scores is used to compute the final *RPComposer* scores. This computation is performed by the "RPComposer" block in Fig. 2.

Definition 4 (*RPComposer Scores*) Consider a training bug report collection BR, and its corresponding classifiers for meta, description, and comments features (Cla_{Meta}, Cla_{Desc}, and $Cla_{Comment}$), respectively. For a new bug report br, we compute its corresponding meta, description and comments scores, then its RPComposer scores, denoted as $Reopen(br, r)$ and $Reopen(br, \bar{r})$, which are linear combinations of the three scores, defined as follows:

$$\begin{aligned} Reopen(br, r) &= \alpha \times Meta(b, r) + \beta \times Desc(b, r) \\ &\quad + \gamma \times Comment(b, r) \qquad (2) \\ Reopen(br, \bar{r}) &= \alpha \times Meta(b, \bar{r}) + \beta \times Desc(b, \bar{r}) \\ &\quad + \gamma \times Comment(b, \bar{r}) \qquad (3) \end{aligned}$$

In the above equations, $\alpha \in [0, 1]$, $\beta \in [0, 1]$, and $\gamma \in [0, 1]$. Based on the above, we define the normalized RPComposer score as:

$$NormReopen(br) = \frac{Reopen(br, r)}{(Reopen(br, r) + Reopen(br, \bar{r}))} \qquad (4)$$

Due to the class imbalance phenomenon, if we directly compare the *RPComposer* scores to decide whether the new bug report br will be reopened (i.e., if $Reopen(br, r) \geq Reopen(br, \bar{r})$, then predict that the report will be reopened; else, predict that it will not be reopened), then the prediction performance can be low. To improve the performance, we propose to predict the class of bug report br, denoted as $Class(br)$, by using a reopened threshold $threshold$, as follows:

$$Class(br) = \begin{cases} r, & if\ NormReopen(br) \geq threshold \\ \bar{r}, & Otherwise \end{cases} \quad (5)$$

The value of $threshold$ can be trained from the training bug report collection. To automatically produce good α, β, γ, and $threshold$ values for *RPComposer*, we propose a greedy algorithm. Algorithm 2 presents the detailed steps to estimate good α, β, γ, and $threshold$ values. We first divide training bug report collection BR into two subsets: BR_1 and BR_2 according to the dividing coefficient sp (Line 10). By default, we set the dividing coefficient as 90 %, and the division process is the same as we do in imbalance feature selection for textual features (see Algorithm 1). Next, we initialize α, β, γ, and $threshold$ values to 0 at Line 11. Then, we build the classifiers (i.e., Cla_{Meta}, Cla_{Desc}, and $Cla_{Comment}$) for meta features, description features, and comment features using BR_1, and compute their corresponding meta, description, and

Algorithm 2 Estimation of Good α, β, γ, and $threshold$ Values in *ReopenPredictor*

1: **Estimatevalue**(BR, sp, $META$,$DESC$,$COMMENT$)
2: **Input:**
3: BR: Training Bug Report Collection
4: sp: Dividing Coefficient
5: $META$: Meta Features for BR
6: $DESC$: Description Features for BR
7: $COMMENT$: Comment Features for BR
8: **Output:** α, β, γ, and $threshold$
9: **Method:**
10: Divide BR into two subsets BR_1 and BR_2 according to sp;
11: α=0, $\beta = 0$, $\gamma = 0$, $threshold = 0$;
12: Build Cla_{meta} for $META$ and BR_1, compute meta scores for each bug report in BR_2;
13: Build Cla_{desc} for $DESC$ and BR_1, compute description scores for each bug report in BR_2;
14: Build $Cla_{comment}$ for $COMMENT$ andBR_1, compute comment scores for each bug report in BR_2;
15: **for all** α from 0 to 1, every time increase α by 0.1 **do**
16: **for all** β from 0 to 1,every time increase β by 0.1 **do**
17: **for all** γ from 0 to 1,every time increase γ by 0.1 **do**
18: **for all** $threshold$ from 0 to 1,every time increase $threshold$ by 0.01 **do**
19: **for all** Bug report br in BR_2 **do**
20: Compute *RPComposer* score according to Definition 4 ;
21: Judge whether br would be reopened by Equation 5;
22: **end for**
23: Evaluate the performance by computing reopened F-measure;
24: **end for**
25: **end for**
26: **end for**
27: **end for**
28: **Return** α, β, γ, and $threshold$ which give the best reopened F-measure

comment scores of bug reports in BR_2 at Lines 12, 13 and 14, respectively. Next, we incrementally increase α, β, γ, and $threshold$ values (Lines 15–18). We increase α, β, and γ values from 0 to 1, in 0.1 increments. We increase the $threshold$ value from 0 to 1, in 0.01 increments. We use a finer granularity step to tune $threshold$ since it directly decides the predicted class label. We use a coarser granularity step to tune α, β, and γ values to reduce the computational cost in the tuning process. For each configuration of α, β, γ, and $threshold$ values, we build a composite model and compute the resultant F-measure score using bug reports in BR_2 (Lines 20–23). Finally, Algorithm 2 returns α, β, γ, and $threshold$ values resulting in the best F-measure (Line 28).

6 Experiments and results

In this section, we evaluate *ReopenPredictor*. The experimental environment is a Windows 7, 64-bit, Intel(R) Xeon(R) 2.53 GHz server with 24 GB RAM. We first present our experiment setup and five research questions (Sects. 6.1 and 6.2). We then present our experiment results that answer the five research questions (Sects. 6.3, 6.4, 6.5 6.6, and 6.7). Next, we discuss important textual features that are related to reopened and non-reopened bug reports (Sect. 6.8). Finally, we describe some threats to validity (Sect. 6.9).

6.1 Experiment setup

For comparison purposes, we reuse the same data used by Shihab et al. (2012) to evaluate their approach. The Eclipse dataset contains bug reports that are reported from 2002-04-10 to 2007-05-29. We do not analyze the entire Eclipse bug reports but only bug reports from the Platform component which are much less in number than the entire Eclipse bug reports. The Apache HTTP dataset contains bug reports that are reported from 2001-01-08 to 2010-04-29. The OpenOffice dataset contains bug reports that are reported from 2000-10-17 to 2010-05-19.

The statistics of the three datasets are presented in Table 2. The rows correspond to the number of total collected bug report, the number of bug reports whose statuses are "resolved", the number of bug reports whose links to code changes or patches

Table 2 Statistics of collected bug reports

	Eclipse	Apache HTTP	OpenOffice
Total collected bug reports	18.312	32,680	106,015
Resolved bug reports	3,903	28,738	86,993
Bug reports linked to commit logs	1,530	14,359	40,169
Reopened bug reports	246	927	10,569
Non-reopened bug reports	1,284	13,432	29,600
Terms in description	2,695	14,710	21,589
Terms in comments	3,593	10,434	24,961

are existed, the number of reopened bug reports, the number of non-reopened bug reports, the number of terms in the description, and the number of terms in comments, respectively. To explain the data in Table 2, we take Apache HTTP as an example. In Apache HTTP, there are in total 32,680 bug reports in its corresponding bug tracking systems, and only 28,738 bug reports are resolved (e.g., closed at least once). Since our meta features need to consider some features related to the bug fix (e.g., number of source code files related to the bug), we need to mine the commit logs to link bug reports and their corresponding fixes. Thus, among the resolved bug reports, there are 14,359 bug report could find the links. Finally, among the 14,359 bug reports, only 927 bug report are opened bug reports, and the remaining 13,432 bug reports are non-reopened.

From Table 2, we also notice the Apache HTTP project dataset has a severe class imbalance problem; the ratio of reopened to non-reopened bug reports is 1:14.5. The ratio of reopened and non-reopened bugs reports for the Eclipse dataset is 1:5.3, while the ratio for the OpenOffice dataset is 1:2.8.

We collect the description, summary, and comment texts from each bug report. We use the WVTool[5] (Wurst 2007) to extract words from these texts. The WVTool is a flexible Java library for statistical language modeling, which is used to create word vector representations of text documents in the vector space model. We use the WVTool to remove stop words, do stemming, and produce "bags of words" from the texts. Words appearing less than 2 times are discarded to remove noisy features.

Stratified tenfold cross validation (Han and Kamber 2006) is used to evaluate the performance of reopened bug prediction. We randomly divide the dataset into tenfolds. Of these tenfolds, ninefolds are use to train the classifier, while the remaining onefold is used to evaluate the performance. The class distribution in the training and testing datasets is kept the same as the original dataset to simulate real-life usage of the algorithm.

To evaluate the performance of our approach, we choose the six metrics used in the first study on reopened bug prediction by Shihab et al. (2010, 2012): reopened precision (Precision(re)), reopened recall (Recall(re)), reopened F-measure ((F-measure(re)), non-reopened precision (Precision(nre)), non-reopened recall (Recall(nre)), and non-reopened F-measure (F-measure(nre)). Reopened (non-reopened) precision refers to the proportion of bug reports that are correctly labeled among those labeled as reopened (non-reopened). Reopened (non-reopened) recall refers to the proportion of reopened (non-reopened) bug reports that are correctly labeled. F-measure is the harmonic mean of precision and recall. It could be viewed as a summary measure that combines both precision and recall—it evaluates if an increase in precision (recall) outweighs a reduction in recall (precision). In this paper, since reopened bug reports is the minority class and we are interested in predicting which bug reports get reopened, the reopened F-measure is the most important evaluation metric.

Past studies have highlighted that it is important for one to use evaluation metrics that are appropriate to the problem at hand (Powers 2011; Jiang et al. 2008). For reopen bug prediction, two factors affect the utility of a bug prediction tool: first, it

[5] http://sourceforge.net/projects/wvtool/

needs to be able to flag a substantial number of the reopened bugs; second, it should not wrongly flag too many bugs as reopened bugs when they are not. Two metrics namely reopened recall and precision capture these factors well. A high recall means that most of the reopened bugs are flagged. A high precision means that the number of wrongly flagged bug reports is low. If recall is too low, developers will not use the tool since most reopened bug reports are missed. If precision is too low, developers will also not use the tool since most prediction results would be wrong. Thus, they are equally important. We use F-measure (i.e., F1) as a summary measure, which is the harmonic mean of precision and recall. We set the parameter β of F-measure equal to 1—this means that precision and recall are given the same weight.

We implement *ReopenPredictor* on top of Weka[6] (Hall et al. 2009). There are three classifiers in our framework: Cla_{Meta}, Cla_{Desc}, and $Cla_{Comment}$. The first classifier classifies meta features. Our previous study investigates the effectiveness of various classification algorithms to predict reopened bug reports using meta features (Xia et al. 2013). A past paper has shown that bagging with C4.5 decision tree performs the best for reopened bug prediction (Xia et al. 2013). Thus we use it to build Cla_{Meta}. The last two classifiers classify textual contents. We use multinomial Naive Bayes, that is known to be fast and yet effective enough to classify these textual contents (McCallum et al. 1998), to train Cla_{Desc} and $Cla_{Comment}$.

For feature selection, we use the default number of selected features, i.e., 1,000, for the Apache HTTP and OpenOffice datasets. For the Eclipse dataset, there are fewer words than the other two datasets. There are only 2,605 and 3,431 words in the description and comment texts, respectively. Thus, to reduce noise, we reduce the number of selected features to 500 for Eclipse.

6.2 Research questions

In this paper, we would like the answer the following research questions:

RQ1 *What is the performance of ReopenPredictor? How much improvement can it achieve over the method proposed in* (Shihab et al. 2012)?

Shihab et al. (2012) propose an approach to predict reopened bug reports. In this research question, we investigate the extent our approach (ReopenPredictor) outperforms the state-of-the-art approach. To answer this research question, we compare reopened precision (Precision(re)), reopened recall (Recall(re)), reopened F-measure ((F-measure(re)), non-reopened precision (Precision(nre)), non-reopened recall (Recall(nre)), and non-reopened F-measure (F-measure(nre)) of ReopenPredictor with those of Shihab et al.'s approach.

RQ2 *Do different numbers of selected features affect the performance of ReopenPredictor?*

By default, we set the number of features as 500 for Eclipse, and 1,000 for Apache HTTPD and OpenOffice. In this research question, we would like to investigate

[6] http://www.cs.waikato.ac.nz/ml/weka/

whether different numbers of selected features affect the performance of ReopenPredictor much. Ideally, since users do not know how to choose the best number of selected features for a new dataset, the performance of ReopenPredictor should be relatively stable for different numbers of selected features, as long as they are within a reasonable range. To answer this research question, we vary the number of selected features from 100 to 2,000, and plot ReopenPredictor's performance for Eclipse, Apache HTTPD and OpenOffice.

RQ3 *Does our proposed imbalanced feature selection method really improve the performance of reopened bug prediction?*

Many past studies on bug report analysis (e.g., severity prediction (Tian et al. 2012b; Menzies and Marcus 2008), bug triaging (Anvik et al. 2006), duplicate bug report detection (Sun et al. 2011), bug linking (Zhou et al. 2012)) do not perform feature selection. We however proposed a new imbalanced feature selection method (see Sect. 4) and apply it to reduce the number of features. In this research question, we would like to investigate whether the proposed imbalanced feature selection method really improves the performance of *ReopenPredictor*. If it does not improve, then our technical contribution is not useful. To answer this research question, we first investigate the performance of imbalanced feature selection method on Cla_{Desc} and $Cla_{Comment}$. Next, we investigate the effect of imbalanced feature selection on the overall framework. We create a variant of *ReopenPredictor* by disabling feature selection and call the resultant technique $ReopenPredictor^{All}$.

RQ4 *Which set of features are the most important to predict reopened bugs?*

In this paper, we use many features, such as meta features, description features, and comment features. In total, we have thousands of features. In this research question, we would like to investigate which features are the most important to predict reopened bugs. We want to perform a deeper analysis on factors that make our approach works. To answer this research question, we extract discriminative features from the thousands of features. We extract the top-20 features per label based on their information gain scores (Han and Kamber 2006).

RQ5 *What is the benefits of our composite classification-based framework?*

Since *ReopenPredictor* combines three classifiers built on three types of features (i.e., meta, description, and comment features), in this research question, we would like to investigate whether the composite classification-based framework could achieve a better performance than a single classifier. The answer to this question would determine whether it is necessary to combine different classifiers to better predict reopened bug reports. To answer this research question, we build a classifier for each of the three types of features, and compare the performance of the single classifiers with *ReopenPredictor*. C4.5 decision tree and naive Bayes (Han and Kamber 2006) are used to build the single classifiers.

6.3 RQ1: performance of ReopenPredictor

Table 3 presents the experiment results of *ReopenPredictor* and the method proposed by Shihab et al. (2012). We notice that the experiment results are a little different

Table 3 Experiment results for *ReopenPredictor* compared with the method proposed by Shihab et al. (2010, 2012)

Project	Algorithm	Prec.(re)	Rec.(re)	F.(re)
Eclipse	*ReopenPredictor*	0.822	0.680	0.744
	Shihab et al.	0.459	0.711	0.558
	Improvement	79.08 %	−4.36 %	33.33 %
Apache HTTP	*ReopenPredictor*	0.799	0.742	0.770
	Shihab et al.	0.538	0.940	0.684
	Improvement	48.51 %	−21.06 %	12.57 %
Openoffice	*ReopenPredictor*	0.858	0.863	0.860
	Shihab et al.	0.782	0.894	0.834
	Improvement	9.72 %	−3.47 %	3.12 %
Project	Algorithm	Prec.(nre)	Rec.(nre)	F.(nre)
Eclipse	*ReopenPredictor*	0.940	0.972	0.956
	Shihab et al.	0.938	0.84	0.886
	Improvement	0.21 %	15.71 %	7.90 %
Apache HTTP	*ReopenPredictor*	0.982	0.987	0.985
	Shihab et al.	0.996	0.944	0.969
	Improvement	−1.41 %	4.56 %	1.65 %
Openoffice	*ReopenPredictor*	0.951	0.949	0.95
	Shihab et al.	0.96	0.911	0.935
	Improvement	−0.94 %	4.17 %	1.60 %

than what are reported in Shihab et al. (2012). This is the case as the tenfold cross validation used in our experiments *randomly* partitions the dataset into 10 sets. Due to the randomness of the process, the resultant sets are different than those produced by the random partitioning performed in Shihab et al.'s experiments.

The reopened F-measure of *ReopenPredictor* are 0.744, 0.770, and 0.860 for Eclipse, Apache HTTP and OpenOffice, respectively. These improve the F-measure of the method proposed by Shihab et al. by 33.33, 12.57 and 3.12 % for Eclipse, Apache HTTP, and OpenOffice, respectively. Averaging across the three datasets, the average improvement achieved by *ReopenPredictor* is 16.34 %. 3 *ReopenPredictor* shows much better performance for the Eclipse dataset. The improvements of reopened precision and F-measure are 79.08 and 33.33 % respectively.

We also notice that the improvements for the OpenOffice dataset is not as substantial as the improvements for the Eclipse and Apache HTTP datasets. The improvement of reopened F-measure for OpenOffice dataset is only 3.12 %. We re-check the class distribution of OpenOffice, and find that the ratio of reopened and non-reopened bug reports is 1:2.8. This is much less imbalanced than the other two datasets, i.e., 1:5.3 and 1:14.5. This indicates that our approach is best suited for highly imbalanced problems, for less imbalanced datasets, it seems that our approach does not achieve as high of an improvement over Shihab et al.'s approach.

Although our precision and recall scores are not perfect, we believe they are reasonably good. Our precision and recall are approximately 0.7–0.8. These numbers mean that out of 10 bug reports that are predicted to be reopened, 7–8 of them are really reopened; also, 70–80 % of bug reports that are eventually reopened are flagged by our tool as such. Many past automated software engineering tools proposed in the literature have precision and recall scores of around 0.7 (e.g., Tian et al. 2012a; Thung et al. 2012; Zhang et al. 2013; Wu et al. 2011; Nguyen et al. 2012) or lower (e.g., Panichella et al. 2014; Canfora et al. 2013; Hooimeijer and Weimer 2007a).

6.4 RQ2: effect of varying the number of selected features

We investigate the effect of the number of the selected features on the performance of *ReopenPredictor*. We vary the number of selected features (i.e., l in Algorithm 1) from 100 to 2,000. We plot the resultant reopened F-measure and other evaluation metrics for Eclipse, Apache HTTP, and OpenOffice datasets in Figs. 3, 4, and 5, respectively.

We notice that reopened F-measure shows stable performance for all the three datasets, i.e., there is little difference with varying number of selected features. This is because in our model building phase, we choose $alpha$, $beta$, $gamma$, and $threshold$ values which maximize the reopened F-measure. We notice that for Eclipse the reopened F-measure scores drops substantially if we select more than 1500 features. This is because there are only 2,605 and 3,431 features in the description and comment texts, respectively. If we choose too many features, then noisy features will be selected too, negatively impacting our F-measure values. For the other two projects, Apache HTTP and OpenOffice, since they have more than 10,000 features, we notice that the reopened F-measure values do not drop substantially when we select more than 1,500 features. And for these two projects, in general, the reopened F-measure values are relatively stable when we vary the number of features from 100 to 2,000.

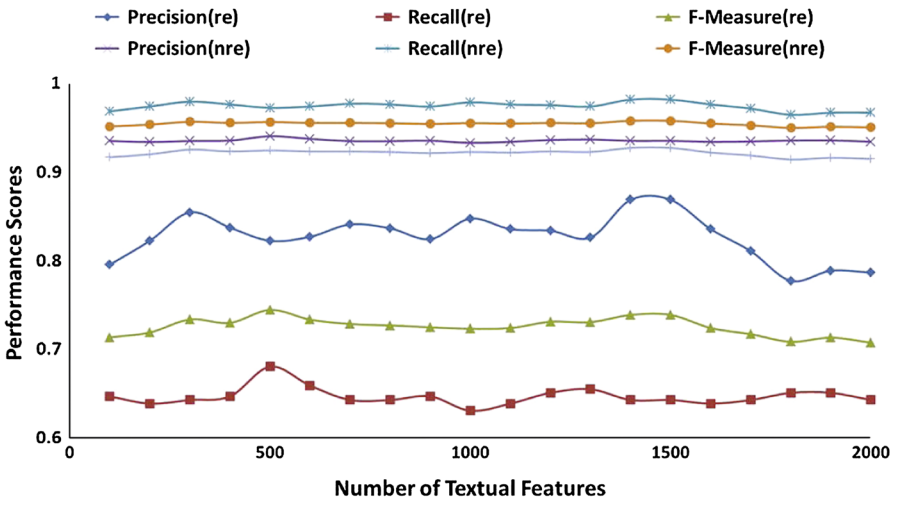


Fig. 3 Experiment results of *ReopenPredictor* for Eclipse with textual features vary from 100 to 2,000

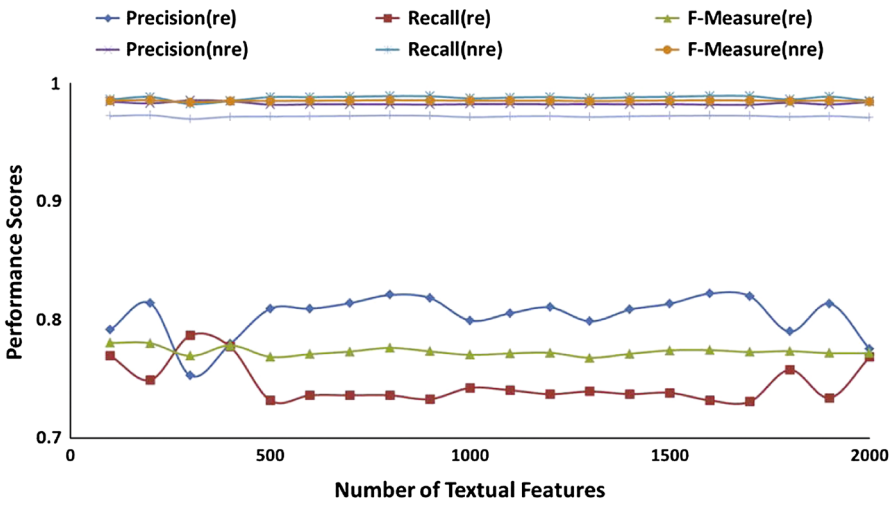


Fig. 4 Experiment results of *ReopenPredictor* for Apache HTTP with textual features vary from 100 to 2,000

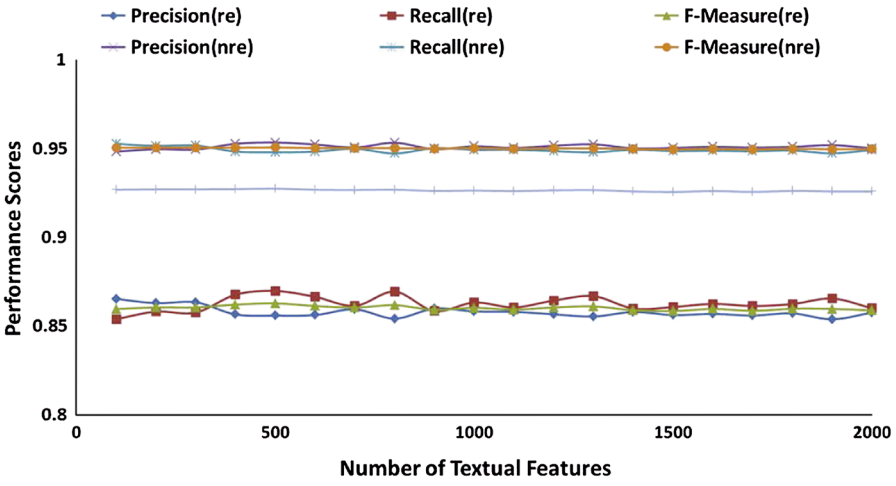


Fig. 5 Experiment results of *ReopenPredictor* for OpenOffice with textual features vary from 100 to 2,000

We notice that reopened precision and recall vary with the different number of selected features. For example, in the Eclipse dataset, when the number of features selected is between 400 and 500, reopened precision substantially increases, while reopened recall decreases. This phenomenon happens in Apache HTTP when the number of features is around 1,600. However, the maximum difference of reopened precision and recall is less then 0.5. So we can still consider our method to be fairly stable.

6.5 RQ3: benefit of imbalanced feature selection

First, we would like to investigate the effect of the imbalanced feature selection on Cla_{Desc}. We train another classifier Cla^{All}_{Desc} that takes all features (i.e., no feature selection is performed). We compare the performance of Cla_{Desc} and Cla^{All}_{Desc}. The result of our comparison is shown in Table 4. We find that Cla_{Desc} outperforms Cla^{All}_{Desc} on the reopened F-measure scores. This shows that feature selection is beneficial at least for description text.

Next, we would like to investigate the effect of imbalanced feature selection on $Cla_{Comment}$. Again, we train another classifier $Cla^{All}_{Comment}$ that takes all features. We compare $Cla_{Comment}$ and $Cla^{All}_{Comment}$. The result is shown in Table 5. We find that $Cla_{Comment}$ outperforms $Cla^{All}_{Comment}$ on the reopened F-measure scores. This shows that feature selection is beneficial, at least for comment text.

Finally, we would like to investigate the effect of imbalanced feature selection on the overall framework. We train another *ReopenPredictor* by disabling all feature selection. We call the resultant technique $ReopenPredictor^{All}$. We compare *ReopenPredictor* score and $ReopenPredictor^{All}$. The result is shown in Table 6. Again, we find that the *ReopenPredictor* score outperforms $ReopenPredictor^{All}$

Table 4 Experiment results of Cla_{Desc} and Cla^{All}_{Desc} for description text

Project	Description	Prec.(re)	Rec.(re)	F.(re)
Eclipse	Cla_{Desc}	0.235	0.154	0.186
	Cla^{All}_{Desc}	0.192	0.121	0.149
	Improvement	22.40 %	27.27 %	24.83 %
Apache HTTP	Cla_{Desc}	0.091	0.040	0.055
	Cla^{All}_{Desc}	0.078	0.033	0.047
	Improvement	17.01 %	19.35 %	18.64 %
Openoffice	Cla_{Desc}	0.454	0.463	0.458
	Cla^{All}_{Desc}	0.438	0.461	0.449
	Improvement	3.78 %	0.29 %	2.05 %
Project	Description	Prec.(nre)	Rec.(nre)	F.(nre)
Eclipse	Cla_{Desc}	0.848	0.903	0.874
	Cla^{All}_{Desc}	0.842	0.901	0.871
	Improvement	0.071	0.22 %	0.34 %
Apache HTTP	Cla_{Desc}	0.936	0.972	0.954
	Cla^{All}_{Desc}	0.936	0.973	0.954
	Improvement	0.00 %	−0.02 %	0.00 %
Openoffice	Cla_{Desc}	0.807	0.802	0.804
	Cla^{All}_{Desc}	0.804	0.788	0.796
	Improvement	0.37 %	1.67 %	1.02 %

Table 5 Experiment results of $Cla_{Comment}$ and $Cla^{All}_{Comment}$ for comments text

Project	Comments	Prec.(re)	Rec.(re)	F.(re)
Eclipse	$Cla_{Comment}$	0.194	0.081	0.115
	$Cla^{All}_{Comment}$	0.196	0.037	0.062
	Improvement	−0.76 %	122.22 %	85.93 %
Apache HTTP	$Cla_{Comment}$	0.133	0.04	0.061
	$Cla^{All}_{Comment}$	0.118	0.029	0.047
	Improvement	11.99 %	37.04 %	31.24 %
Openoffice	$Cla_{Comment}$	0.504	0.266	0.348
	$Cla^{All}_{Comment}$	0.507	0.237	0.323
	Improvement	−0.57 %	12.42 %	7.93 %
Project	Comments	Prec.(nre)	Rec.(nre)	F.(nre)
Eclipse	$Cla_{Comment}$	0.842	0.935	0.886
	$Cla^{All}_{Comment}$	0.840	0.971	0.901
	Improvement	0.16 %	−3.69 %	−1.66 %
Apache HTTP	$Cla_{Comment}$	0.937	0.982	0.959
	$Cla^{All}_{Comment}$	0.936	0.985	0.960
	Improvement	0.05 %	−0.31 %	−0.13 %
Openoffice	$Cla_{Comment}$	0.776	0.906	0.836
	$Cla^{All}_{Comment}$	0.771	0.918	0.838
	Improvement	−0.57 %	−1.23 %	−0.24 %

on the reopened F-measure scores. This shows that feature selection is beneficial for $ReopenPredictor$ as a whole.

To further evaluate whether the difference between *ReopenPredictor* and $ReopenPredictor^{All}$ is significant, we repeat the tenfold cross validation 100 times, and record the reopened F-measure for each time. In total, we have 100 paired data points (reopened F-measures). Next, we perform a Wilcoxon signed rank test (Wilcoxon, 1945) on the 100 paired data, and record the p value. For Eclipse, we notice the difference is significant with p value of 0.007515. For Apache HTTP, the difference is significant with p value of 0.03772. For OpenOffice, the difference is significant with p value of less than 0.001.

6.6 RQ4: important features of *ReopenPredictor*

Table 7 presents the top-20 most discriminative features in Eclipse, Apache HTTP, and OpenOffice. We notice that some meta features (i.e., comments text, description text, fixer name, comments size, report name, time days, component) appear in the three projects. Aside from these meta features, some textual features, corresponding to stemmed words that appear in bug reports, are also good indicators to reopened bug

Table 6 Experiment results of *ReopenPredictor* compared with $ReopenPredictor^{all}$

Project	Algorithm	Prec.(re)	Rec.(re)	F.(re)
Eclipse	$ReopenPredictor$	0.822	0.680	0.744
	$ReopenPredictor^{All}$	0.608	0.633	0.620
	Improvement	35.20 %	7.42 %	20.00 %
Apache HTTP	$ReopenPredictor$	0.799	0.742	0.770
	$ReopenPredictor^{All}$	0.655	0.849	0.739
	Improvement	22.03 %	−12.60 %	4.15 %
Openoffice	$ReopenPredictor$	0.858	0.863	0.860
	$ReopenPredictor^{All}$	0.848	0.857	0.853
	Improvement	1.18 %	0.70 %	0.82 %
Project	Algorithm	Prec.(nre)	Rec.(nre)	F.(nre)
Eclipse	$ReopenPredictor$	0.940	0.972	0.956
	$ReopenPredictor^{All}$	0.929	0.922	0.925
	Improvement	1.18 %	5.42 %	3.35 %
Apache HTTP	$ReopenPredictor$	0.982	0.987	0.985
	$ReopenPredictor^{All}$	0.989	0.969	0.979
	Improvement	−0.74 %	1.85 %	0.60 %
Openoffice	$ReopenPredictor$	0.951	0.949	0.95
	$ReopenPredictor^{All}$	0.949	0.945	0.947
	Improvement	0.21 %	0.42 %	0.32 %

prediction. For example, the term “verif” in the comments text is a good indicator to predict whether a bug would get reopened in Eclipse and OpenOffice, which represent that a developer is confident that this bug is already fixed, and “verified”.[7]

6.7 RQ5: benefits of composite classification-based framework

Tables 8 and 9 present the experiment of *ReopenPredictor* compared with C4.5 and naive Bayes, respectively. Again, we find that the *ReopenPredictor* score outperforms C4.5 and Naive Bayes on the reopened F-measure scores. This shows that the benefits of combining the different classifiers. *ReopenPredictor* improves the F-measure of the C4.5 by 177.45, 9274.75 and 45.60 % for Eclipse, Apache HTTP, and OpenOffice, respectively. *ReopenPredictor* improves the F-measure of the naive Bayes by 150.92, 413.85 and 52.25 % for Eclipse, Apache HTTP, and OpenOffice, respectively.

[7] For more description of the terms in description and comments, please refer to Sect. 6.8.3.

Table 7 Top-20 discriminative features based on information gain scores

Eclipse		Apache		OpenOffice	
Features	Scores	Features	Scores	Features	Scores
Comments text	0.253	Last status	0.200	Comments text	0.248
Description text	0.224	Description text	0.108	Fixer exp.	0.184
C "verif"	0.059	Comments text	0.066	Last status	0.157
Fixer name	0.023	Fixer Name	0.056	**C "verif"**	0.126
Comments Size	0.020	Comments Size	0.030	Description text	0.119
Reporter Name	0.018	No. of comments	0.027	Reporter Name	0.101
Time Days	0.014	Component	0.022	No. of comments	0.087
Component	0.009	Reporter Name	0.019	**C "fic"**	0.080
D "readanddispatch"	0.007	Time Days	0.017	Comments Size	0.080
C "unfortun"	0.007	No. of source files	0.007	Fixer name	0.077
D "programat"	0.007	Platform	0.006	Day of year	0.043
Platform	0.006	Severity	0.005	No. of source files	0.041
C "rel"	0.006	Severity changed	0.004	Component	0.030
D "runeventloop"	0.006	Fixer exp.	0.003	Time Days	0.028
D "ideapplic"	0.006	description size	0.003	Weekday	0.027
D "sendev"	0.006	**C "problem"**	0.002	Reporter exp.	0.024
D "event"	0.005	**C "pleas"**	0.002	**C "pleas"**	0.020
C "def"	0.005	**C "fil"**	0.002	month	0.018

Text features in description and comments are bold *C* comments, *D* description

6.8 Discussion

6.8.1 Longitudinal data setup

To investigate whether our tool can be used to solve the problem in the same setting as the one in practice, we performed an experiment using a longitudinal data setup following Tamrawi et al. (2011) and Bhattacharya and Neamtiu (2010). We sorted the bug reports in the order they are received and split them into 11 non-overlapping windows of equal sizes. The process then proceeds as follows: First, in fold 1, we train using bug reports in window 0, and test the trained model using the bug reports in window 1. Then, in fold 2, we train using bug reports in windows 0 and 1, and test the trained model using the bug reports in window 2, and so on. We proceed in a similar manner for the next folds. In the final fold (i.e., fold 10), we train using bug reports in window 0–9, and test using the bug reports in window 10. We record the average performance across the tenfolds. If there are more distinct terms, there are more topics. In default, we set the number of terms in imbalance feature selection to 15 % of the number of distinct terms in the training data. We found that the results remain the same, i.e., we can achieve precision and recall scores of approximately 0.7 or higher, our approach also outperforms the prior work by Shihab et al.

Table 8 Experiment results of *ReopenPredictor* compared with C4.5

Project	Algorithm	Prec.(re)	Rec.(re)	F.(re)
Eclipse	*ReopenPredictor*	0.822	0.680	0.744
	C4.5	0.429	0.195	0.268
	Improvement	91.80 %	248.50 %	177.45 %
Apache HTTP	*ReopenPredictor*	0.799	0.742	0.770
	C4.5	0.085	0.004	0.008
	Improvement	838.83 %	17905.85 %	9274.75 %
Openoffice	*ReopenPredictor*	0.858	0.863	0.860
	C4.5	0.648	0.543	0.591
	Improvement	32.48 %	58.96 %	45.60 %
Project	Algorithm	Prec.(nre)	Rec.(nre)	F.(nre)
Eclipse	*ReopenPredictor*	0.940	0.972	0.956
	C4.5	0.860	0.950	0.903
	Improvement	9.26 %	2.30 %	5.87 %
Apache HTTP	*ReopenPredictor*	0.982	0.987	0.985
	C4.5	0.936	0.997	0.965
	Improvement	4.97 %	−0.98 %	2.05 %
Openoffice	*ReopenPredictor*	0.951	0.949	0.95
	C4.5	0.846	0.895	0.869
	Improvement	12.45 %	6.09 %	9.27 %

Table 10 presents the experiment results of *ReopenPredictor* and the method proposed by Shihab et al. (2012) with the longitudinal data setup. The reopened F-measure of *ReopenPredictor* are 0.685, 0.750, and 0.687 for Eclipse, Apache HTTP and OpenOffice, respectively. These values are an improvement over the F-measure reported by Shihab et al. by 14.57, 9.06 and 4.97 % for Eclipse, Apache HTTP, and OpenOffice, respectively. Averaging across the three datasets, the average improvement achieved by *ReopenPredictor* is 9.53 %.

We also investigate the performance of the *ReopenPredictor* with the number of features varying from 1 to 20 % of the unique terms in the training datasets. We plot the reopened F-measure and other evaluation metrics for Eclipse, Apache HTTP, and OpenOffice datasets in Figs. 6, 7, and 8, respectively. We notice the difference between the different number of terms are minor, indicating that the number of terms does not have a large impct on the F-measure.

6.8.2 Periodic update

It is possible that word frequency in bug reports changes over time; some words that are indicative of reopened bugs might no longer be indicative in the future. To deal with this possibility, our approach can be retrained periodically using bug reports that

Table 9 Experiment results of *ReopenPredictor* compared with Naive Bayes

Project	Algorithm	Prec.(re)	Rec.(re)	F.(re)
Eclipse	*ReopenPredictor*	0.822	0.680	0.744
	Naive Bayes	0.226	0.431	0.297
	Improvement	263.70 %	57.81 %	150.92 %
Apache HTTP	*ReopenPredictor*	0.799	0.742	0.770
	Naive Bayes	0.100	0.296	0.150
	Improvement	696.08 %	151.03 %	413.85 %
Openoffice	*ReopenPredictor*	0.858	0.863	0.860
	Naive Bayes	0.463	0.725	0.565
	Improvement	85.37 %	19.10 %	52.25 %
Project	Algorithm	Prec.(nre)	Rec.(nre)	F.(nre)
Eclipse	*ReopenPredictor*	0.940	0.972	0.956
	Naive Bayes	0.868	0.717	0.786
	Improvement	8.29 %	35.51 %	21.71 %
Apache HTTP	*ReopenPredictor*	0.982	0.987	0.985
	Naive Bayes	0.944	0.817	0.876
	Improvement	4.04 %	20.79 %	12.45 %
Openoffice	*ReopenPredictor*	0.951	0.949	0.95
	Naive Bayes	0.877	0.700	0.778
	Improvement	8.47 %	35.62 %	22.06 %

are reported and resolved recently, whenever the accuracy of our technique degrades in predicting reopened bugs.

6.8.3 Qualitative analysis

Here, we want to perform a qualitative analysis of why our new feature selection technique worked in RQ3. Our feature selection technique ranks description and comment features in terms of their correlation coefficient (CC) scores as described in Sect. 4. To understand why our feature selection technique works, we first show the features with the highest correlation coefficient scores.

Tables 11, 12 and 13 present the top-20 features in the description and comment texts which are indicative of reopened and non-reopened bug reports in Eclipse, Apache HTTP and OpenOffice projects, respectively. These features are stemmed words from the description and comment texts. Some of the words that appear in the lists indicate particular parts of a software system or features (e.g., "toolitem", "toolbarmanager", "buffermanager", "thead", "toolbar", "databas", and "spreadsheetml") that are hard to fix and correlate with bug reports being reopened. Other words indicate parts of a software system or features (e.g., "structuredviewer", "dialog", "xmlgraph", "viewbox", and "patch") that are easier to fix and correlate with bug reports being not reopened.

Table 10 Experiment results for *ReopenPredictor* compared with the method proposed by Shihab et al. (2010, 2012) with Longitudinal Data Setup

Project	Algorithm	Prec.(re)	Rec.(re)	F.(re)
Eclipse	*ReopenPredictor*	0.734	0.668	0.685
	Shihab et al.	0.547	0.684	0.598
	Improvement	34.33 %	−2.41 %	14.57 %
Apache HTTP	*ReopenPredictor*	0.742	0.770	0.750
	Shihab et al.	0.627	0.781	0.687
	Improvement	18.38 %	−1.33 %	9.06 %
Openoffice	*ReopenPredictor*	0.728	0.786	0.752
	Shihab et al.	0.651	0.820	0.716
	Improvement	11.82 %	−4.15 %	4.97 %
Project	Algorithm	Prec.(nre)	Rec.(nre)	F.(nre)
Eclipse	*ReopenPredictor*	0.936	0.943	0.938
	Shihab et al.	0.935	0.895	0.914
	Improvement	0.07 %	5.35 %	2.62 %
Apache HTTP	*ReopenPredictor*	0.984	0.979	0.982
	Shihab et al.	0.984	0.965	0.974
	Improvement	0.00 %	1.65 %	0.74 %
Openoffice	*ReopenPredictor*	0.943	0.908	0.924
	Shihab et al.	0.939	0.875	0.905
	Improvement	0.39 %	3.76 %	2.12 %

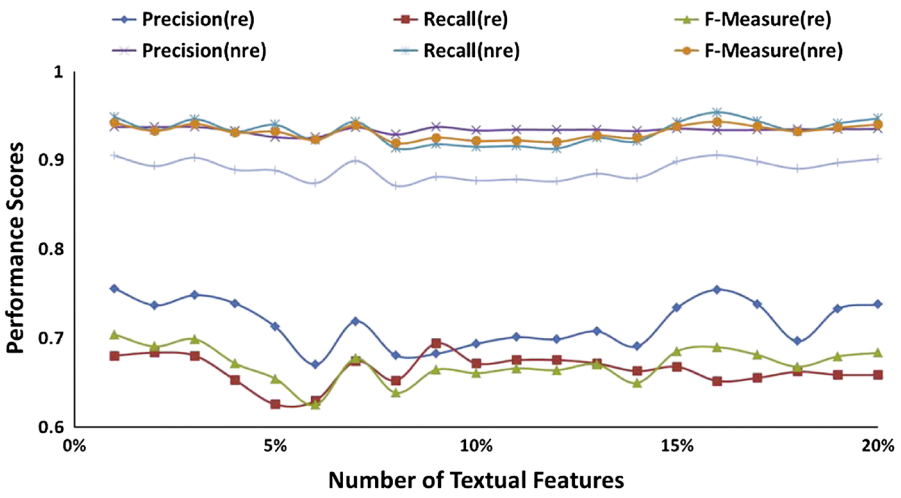


Fig. 6 Experiment results of *ReopenPredictor* for Eclipse with textual features vary from 1 to 20 %

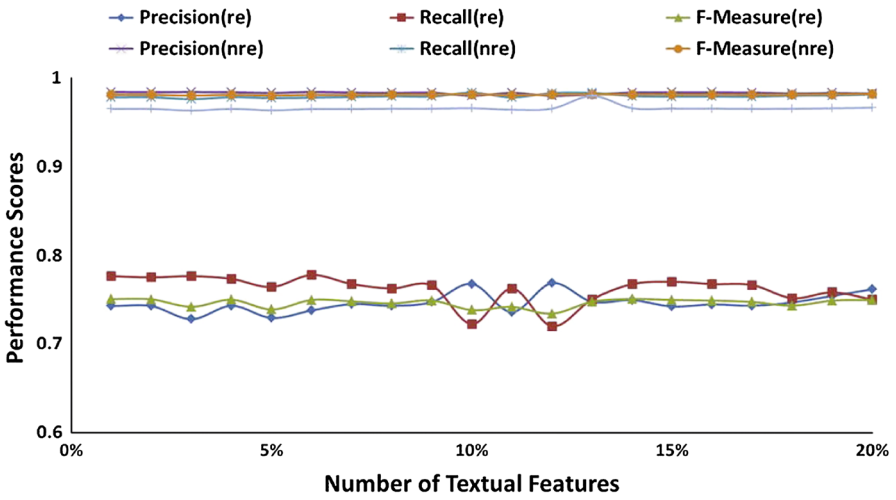


Fig. 7 Experiment results of *ReopenPredictor* for Apache HTTP with textual features vary from 1 to 20 %

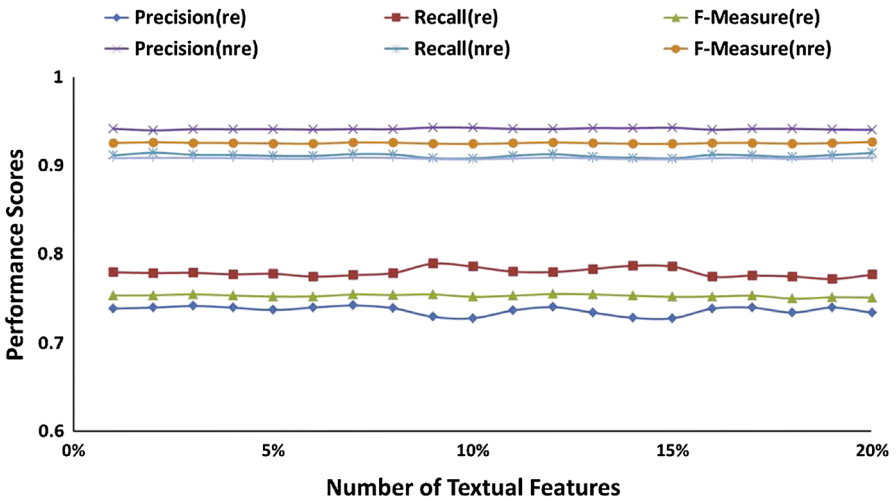


Fig. 8 Experiment results of *ReopenPredictor* for OpenOffice with textual features vary from 1 to 20 %

Although, we do not employ sentiment analysis, some of the words in the list that are related to non-reopened bug reports include: "thank", "great", and "approv" which are strong affirmative words. Similarly, some of the words in the list that are related to reopened bug reports include: "wonder", "unusual", and "problem" which are negative words that indicate doubts on the quality of a bug fix.

Figure 9 presents a bug report from OpenOffice. This bug report is wrongly classified by Shihab et al.'s method (i.e., predicted it as non-reopened). However, our *ReopenPredictor* predicts it as reopened. We manually check the output of each of the three classifiers in *ReopenPredictor*, and we find the description classifier has strong

Table 11 Top-20 textual features (in terms of their CC scores) in the description and comment texts related to reopened and non-reopened bug reports in Eclipse dataset

Description		Comments	
Reopened	Non-reopened	Reopened	Non-reopened
programat	readanddispatch	unusu	verif
horn	runeventloop	helper	unfortun
strok	event	intrus	clos
exact	ideapplic	rel	oper
whatsoever	sendev	def	detect
getactivep	displa	middl	replac
tweak	handleev	japan	thank
wonder	adapt	workaround	defin
dirk	basicrun	sens	inspect
recreat	rundeferredevent	contain	michael
toolitem	nullpointerexcept	creates	hous
toolbarmanager	jfac	accomod	pardon
coolbarmanager	nat	isregister	spam
stand	shar	getedit	attach
advant	structuredviewer	doctyp	rep
viol	runu	bens	great
lif	createandrunwork	alph	approv
optimizeit	eclipsestarter	chief	dialog
divider	platformactiv	unfic	content
iworkbenchpart	upd	unintens	earl

confidence that this bug report will be reopened. And we further manually check the description of the bug report; we notice several terms such as “crash”, “loading”, “slide”, “insert”, “start”, “impress”, “presentation”, “object” which are highly related to reopened bug reports as shown in Table 13.

Figure 10 presents a bug report from Apache. This bug report is also wrongly classified by Shihab et al.’s method (i.e., predicted it as non-reopened). However, our *ReopenPredictor* predicts it as reopened. We manually check the output of each of the three classifiers in *ReopenPredictor*, and we find the comment classifier has strong confidence that this bug report will be reopened. And we further manually check the comments of the bug report; we notice several terms such as “problem”, “Test”, “causes”, “file”, and “make” are highly related to reopened bug reports as shown in Table 12.

6.9 Threats to validity

Threats to internal validity relates to bias and errors in our experiments. The datasets used in our experiments are the same as those used in previous studies (Shihab et al.

Table 12 Top-20 textual features (in terms of their CC Scores) in the description and comment texts related to reopened and non-reopened bug reports in Apache HTTP dataset

Description		Comments	
Reopened	Non-reopened	Reopened	Non-reopened
arent	batik	problem	bulk
leur	docum	pleas	linger
conträ	bugrat	window	overrid
utiliser	transform	caus	viewvc
filtr	patch	loadonstartup	renderer
porté	font	hog	fallback
tomcat	wasn	week	thank
runexecut	improv	fil	xmlgraph
cupé	form	experi	incompat
load	height	mak	hssfcel
alot	ident	run	procedur
pcreposic	rect	error	evalu
processcallback	bridg	accesscontrol	member
removeelementat	howt	applic	accis
buffermanager	usecas	reproduc	javadoc
userbas	viewbox	tomcat	ton
sysde	eval	waitfor	cek
thead	xindic	test	stefan
reaper	href	antivirus	howt
screennam	el	dechunk	vinc

2010, 2012). We have also double checked the datasets and our experiments, still there could be errors that we did not notice. Moreover, since we use textual information in the comments of bug reports, we have double checked that the textual information that we have collected are those available before the bug reports are reopened.

Threats to external validity relates to the generalizability of our results. We have analyzed 56,058 bug reports from three software systems. And in total, we collected 157,007 bug reports, and our datasets takes 35.7 % number of the total collected bug reports. Next, analyzing a substantial proportion of bug reports in selected projects is important for the generalizability of the findings. In this work, we have investigated 8.36 % of bug reports of the Platform component of Eclipse version 3.0, 43.94 % of bug reports of Apache HTTP, and 37.89 % of bug reports of OpenOffice. We believe that we have investigated a substantial proportion of bug reports from these projects. Past studies also only investigate similar number of bug reports from these projects (e.g., Hooimeijer and Weimer 2007a; Zaman et al. 2011; Bhattacharya and Neamtiu 2010). In the future, we plan to reduce this threat to external validity by investigating more bug reports from these projects.

Table 13 Top-20 textual features (in terms of their CC scores) in the description and comment texts related to reopened and non-reopened bug reports in OpenOffice dataset

Description		Comments	
Re-opened	Not re-opened	Re-opened	Not re-opened
bugdoc	transl	bugdoc	verif
impres	openoffic	swqbugfic	fic
sl	build	accept	pleas
oas	sophi	workload	build
docum	download	spreadsheetml	issu
eform	patch	loop	thank
load	thank	naddloc	chang
start	dmak	retarges	openoffic
sorter	project	pri	commis
insers	http	mmahes	check
wizard	upd	jmarm	fil
pres	lou	mmahes	http
crash	autom	eik	sourc
toolbar	cod	overbal	test
assers	solenv	natn	vers
databas	solver	oasisbf	intern
object	instal	anim	targes
dbwizard	kind	timezon	patch
dock	inclus	onload	master
custom	link	colormap	servic

Summary: **Crash** when **loading** another **slide** and switching to custom animation
Description: I **start** a new **impress** and use **insert**/file.
I choose a former produced **presentation** with two **slides**, each containing an **object** with a custom animtion.
I insert one **slide** of them and switch to custom animations: **Crash**.
The ID of the error report is rqvh68.

Fig. 9 Bug report #53696 of OpenOffice. The terms which appears in Table 13 are *underlined*

Threats to construct validity refers to the suitability of our evaluation measures. We use F-measure score as the main evaluation metric which is also used by past studies to evaluate the effectiveness of various prediction algorithms in solving various software engineering tasks (e.g., Nguyen et al. 2012; Peters and Menzies 2012; Tian et al. 2012a, b; Kim et al. 2008; Nam et al. 2013). Thus, we believe there is little threat to construct validity.

Summary: cloneSheet breaks autofilters
Comments1: I could **reproduce** this bug fairly easily. One slight difference however, was that my Excel(2007) completely crashes when I try to sort using the autofilter in the newly copied sheet.
To help diagnose, I cloned the sheet in the input spreadsheet using Excel and ran BiffViewer on that **file**, and the POI generated file.
The first **problem** I noticed was a missing NameRecord (of the 'built-in' kind).
Another cosmetic **problem** was POI's choice of name for the new sheet. Excel adds a space before the bracket e.g. "**Test** Clone (2)", which upsets the file offsets and **causes** false deltas when comparing the BiffViewer files. If fixed both these problems in svn r691740. No junit for the built-in name record yet.
After that fix the next difference I tried working on was within the MSODRAWING record (in the second, copied sheet). Unfortunately, POI does not interpret that record when reading from and existing file (BiffViewer does though), so I had to resort to hex-editing the POI generated XLS file. After changing a few bytes (to **make** the POI generated second MSODRAWING record the same as the Excel generated one), the file loaded OK in Excel, and the autofilters worked fine.
I will attach the BiffViewer dump of the MSODRAWING records before and after manual hex editing.

Fig. 10 Bug report #45720 of Apache HTTP. The terms which appears in Table 12 are *underlined*

7 Related work

In this section, we briefly review related studies. We first compare our work to work on reopened bugs and then we contrast our work with work on managing bug reports.

7.1 Reopened bug reports

To our best knowledge, there are very few prior studies that analyzed reopened bug reports (Shihab et al. 2010, 2012; Zimmermann et al. 2012). Shihab et al. (2010, 2012) propose the problem of identifying reopened bug reports in three open source projects using machine learning algorithms. Zimmermann et al. (2012) analyze and categorize reopened bugs in the Microsoft Windows operating system. The main focus of their study was to investigate the different reasons for bug reopenings.

Our work complements and differs from the aforementioned work in several ways. First, our work extends the work by Shihab et al. (2010, 2012) by improving its performance. Second, our work proposes algorithms that can be used to automatically calculate thresholds needed to optima the performance of *ReopenPredictor*. Furthermore, our work differs from the work by Zimmermann et al. (2012) since our work studies open source data. However, if commercial data is made available we are confident that *ReopenPredictor* can help improve the performance of reopened bugs in commercial projects as well since we believe that effective use of textual information and the imbalanced data phenomenon also exist in commercial reopened bugs.

7.2 Bug report management

There exist machine learning and information retrieval approaches for automatic bug triaging (Anvik et al. 2006; Anvik and Murphy 2007; Tamrawi et al. 2011; Jeong et al. 2009; Matter et al. 2009). The textual description and structure of bug reports from bug tracking systems such as Bugzilla provide a lot of information for bug triaging. Anvik et al. (2006) and Cubranic et al. (2004)used Naive Bayes, SVM, and C4.8 for bug triaging; title, description and summary fields are extracted from bug reports to train their classifier.

Jeong et al. (2009) investigate bug reassignment in some open source software communities, and propose the usage of bug tossing graph to improve bug triaging performance. Bhattacharya and Neamtiu (2010) improve the accuracy of the approach by Jeong et al. further. Tamrawi et al. (2011) propose a method called Bugzie, which uses a fuzzy set and cache-based approach to increase the accuracy of bug triaging.

A number of approaches have been proposed to automatically infer or to help developers in inferring bug reports that are duplicates of one another. Runeson et al. (2007) measure the similarity of two bug reports using $cosine$, $dice$, and $jaccard$ similarity of the term features appearing in the bug reports. The similarity scores are used to identify bug reports that are likely to be duplicate of one another. Wang et al. (2008) propose a duplicate bug report detection approach that integrates bug reports with execution traces. They find that if execution trace information is available duplicate bug reports could be identified with a higher accuracy. Sun et al. (2010) propose a discriminative model based approach using Support Vector Machine to identify bug reports that are duplicate of one another. In a latter work, Sun et al. (2011) extend BM25, a popular information retrieval approach, for duplicate bug report detection.

There are also a number of studies that predict the severity labels of bug reports. Severis, proposed by Menzies and Marcus (2008), performs multi-class classification to predict the five severity labels of bug reports in NASA. Their work is extended by Lamkanfi et al. (2010) which predict two severity labels (severe vs. not severe) of bug reports in a number of Bugzilla bug tracking systems of open source programs. Lamkanfi et al. (2011) also investigate the effectiveness of a number of classification algorithms to predict the severity of bug reports. Our work complements the above studies; after the priority of a bug report could be determined, our approach could be employed to recommend a suitable developer to fix the bug.

Huang et al. (2011) propose a machine learning approach that predict the categories of bug reports based on their impact; the category labels include: capability, security, performance, reliability, requirement, and usability. Gegick et al. (2010) perform text mining to recover security bug reports. Francis et al. (2004) and Pordguski et al. (2003) group reported software failures together based on the similarities of execution traces.

Our work differs from the prior work on bug repository mining in several ways. First, our focus is on reopened bugs. Second, whereas most prior studies used numerical features (some did use textual features also), to the best of our knowledge, our work is one of the first to investigate the use textual information with such detail. We strongly believe that many of our techniques can be used in other contexts as well. In fact, in the future, we plan to investigate how our techniques can be used to improved other

problems such as bug triaging, bug reassignment, bug duplicate detection and bug classification.

7.3 Empirical studies on bug tracking systems

A number of empirical studies have been performed on bug tracking systems. A study on bug report networks in a large open source development community was performed by Sandusky et al. (2004). Anvik et al. (2005) study the characteristics of bug tracking systems of Eclipse and Firefox projects. Bettenburg et al. (2008) characterize what makes a good bug report by surveying a number of Eclipse, Mozilla, and Apache developers. Hooimeijer and Weimer (2007b) develop a model that predicts the quality of bug reports. Anvik et al. (2005) empirically study the characteristics of bug repositories and show findings on the number of reports that a person submits and the proportion of different resolutions.

8 Conclusions and future work

In this paper, we propose *ReopenPredictor* to predict reopened bugs. We first consider more features from bug reports (i.e., meta, description, and comment features), and propose an imbalanced feature selection method to choose the most substantial textual features from both reopened and non-reopened bug reports. Then, *ReopenPredictor* use all of these features, and automatically assign different classifiers with different weights to improve the overall performance. Experiment results show that our *ReopenPredictor* show substantially better performance than the state-of-the-art method proposed by Shihab et al. (2010, 2012). The reopened F-measure are 0.744, 0.770, and 0.860 for Eclipse, Apache HTTP and OpenOffice, respectively, and we improve the reopened F-measure of the method proposed by Shihab et al. by 33.33, 12.57 and 3.12 % for Eclipse, Apache HTTP and OpenOffice, respectively. Averaging across the three datasets, the average improvement achieved by *ReopenPredictor* is 16.34 %.

In the future, we plan to evaluate our *ReopenPredictor* with more reopened bug reports, and develop a better technique which could improve reopened bug report prediction further, in particular, to increase the reopened F-measure. We also plan to experience with more imbalanced learning algorithms (He and Garcia 2009) in our proposed overall framework for reopened bug prediction, for example, SMOTE (Chawla et al. 2002), imbalanced SVM (Akbani et al. 2004), etc. We also plan to investigate sentiment analysis (e.g., Pang and Lee 2008; Liu 2012) tools to improve the effectiveness of ReopenPredictor further.

Acknowledgments This research is sponsored in part by NSFC Program (No. 61103032) and National Key Technology R&D Program of the Ministry of Science and Technology of China (No. 2013BAH01B01). The code can be download from: https://github.com/xinxia1986/reopenBug.

References

Akbani, R., Kwek, S., Japkowicz, N.: Applying support vector machines to imbalanced datasets. Eur. Conf. Mach. Learn. **2004**, 39–50 (2004)

Anvik, J., Murphy, G.: Determining implementation expertise from bug reports. In: MSR (2007)
Anvik, J., Hiew, L., Murphy, G.C.: Coping with an open bug repository. In: ETX, pp. 35–39 (2005)
Anvik, J., Hiew, L., Murphy, G.C.: Who should fix this bug? In: ICSE, pp. 361–370. ACM, New York (2006)
Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)
Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., Zimmermann, T.: What makes a good bug report? In: FSE, pp. 308–318 (2008)
Bhattacharya, P., Neamtiu, I.: Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In: ICSM, pp. 1–10 (2010)
Canfora, G., De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A., Panichella, S.: Multi-objective cross-project defect prediction. In: IEEE Sixth International Conference on Software Testing, Verification and Validation (ICST), 2013, pp. 252–261. IEEE (2013)
Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: synthetic minority over-sampling technique. J. Artif. Intell. Res. **16**, 321–357 (2002)
Čubranić, D.: Automatic bug triage using text categorization. In: SEKE (2004)
Francis, P., Leon, D., Minch, M.: Tree-based methods for classifying software failures. In: ISSRE, pp. 451–462 (2004)
Gegick, M., Rotella, P., Xie, T.: Identifying security bug reports via text mining: an industrial case study. In: MSR, pp. 11–20 (2010)
Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The weka data mining software: an update. SIGKDD Explor. **11**(1), 10–18 (2009)
Han, J., Kamber, M.: Data mining: concepts and techniques. Morgan Kaufmann, San Francisco (2006)
He, H., Garcia, E.: Learning from imbalanced data. Trans. Knowl. Data Eng. **21**(9), 1263–1284 (2009)
Hooimeijer, P., Weimer, W.: Modeling bug report quality. In: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, pp. 34–43. ACM, New York (2007a)
Hooimeijer, P., Weimer, W.: Modeling bug report quality. In: ASE, pp. 34–43 (2007b)
Huang, L., Ng, V., Persing, I., Geng, R., Bai, X., Tian, J.: AutoODC: Automated generation of orthogonal defect classifications. In: ASE, pp. 412–415 (2011)
Jeong, G., Kim, S., Zimmermann, T.: Improving bug triage with bug tossing graphs. In: ESEC/FSE, pp. 111–120 (2009)
Jiang, Y., Cukic, B., Ma, Y.: Techniques for evaluating fault prediction models. Empir. Softw. Eng. **13**(5), 561–595 (2008)
Kim, S., Whitehead, E.J., Zhang, Y.: Classifying software changes: clean or buggy? IEEE Trans. Softw. Eng. **34**(2), 181–196 (2008)
Lamkanfi, A., Demeyer, S., Giger, E., Goethals, B.: Predicting the severity of a reported bug. In: MSR, pp. 1–10 (2010)
Lamkanfi, A., Demeyer, S., Soetens, Q., Verdonck, T.: Comparing mining algorithms for predicting the severity of a reported bug. In: CSMR, pp. 249–258 (2011)
Liu, B.: Sentiment analysis and opinion mining. Synth. Lect. Hum. Lang. Technol. **5**(1), 1–167 (2012)
Matter, D., Kuhn, A., Nierstrasz, O.: Assigning bug reports using a vocabulary-based expertise model of developers. In: MSR, pp. 131–140 (2009)
McCallum, A., Nigam, K., et al.: A comparison of event models for naive bayes text classification. In: AAAI-98 Workshop, Citeseer, vol. 752, pp. 41–48 (1998)
Menzies, T., Marcus, A.: Automated severity assessment of software defect reports. In: ICSM, pp. 346–355 (2008)
Nam, J., Pan, S.J., Kim, S.: Transfer defect learning. In: ICSE, pp. 382–391. IEEE (2013)
Nguyen, A.T., Nguyen, T.T., Nguyen, H.A., Nguyen, T.N.: Multi-layered approach for recovering links between bug reports and fixes. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, p. 63. ACM, New York (2012)
Pang, B., Lee, L.: Opinion mining and sentiment analysis. Found. Trends Inf. Retr. **2**(1–2), 1–135 (2008)
Panichella, A., Oliveto, R., De Lucia, A.: Cross-project defect prediction models: L'union fait la force. In: Software Evolution Week—IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014, pp. 164–173 (2014)
Peters, F., Menzies, T.: Privacy and utility for defect prediction: Experiments with morph. In: ICSE, pp. 189–199. IEEE (2012)

Podgurski, A., Leon, D., Francis, P., Masri, W., Minch, M., Sun, J., Wang, B.: Automated support for classifying software failure reports. In: ICSE, pp. 465–475 (2003)
Powers, D.M.: Evaluation: from precision, recall and f-measure to roc, informedness, markedness & correlation. J. Mach. Learn. Technol. **2**(1), 37–63 (2011)
Runeson, P., Alexandersson, M., Nyholm, O.: Detection of Duplicate Defect Reports Using Natural Language Processing. In: ICSE, pp. 499–510 (2007)
Sandusky, R.J., Gasser, L., Ripoche, G.: Bug report networks: Varieties, strategies, and impacts in a f/oss development community. In: MSR, Citeseer (2004)
Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W., Ohira, M., Adams, B., Hassan, A., Matsumoto, K.: Predicting re-opened bugs: A case study on the eclipse project. In: WCRE, Citeseer, pp. 249–258 (2010)
Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W.M., Ohira, M., Adams, B., Hassan, A.E., ichi Matsumoto, K.: Studying re-opened bugs in open source software. In: Empirical Software Engineering, pp. 1–38 (2012)
Sun, C., Lo, D., Wang, X., Jiang, J., Khoo, S.C.: A discriminative model approach for accurate duplicate bug report retrieval. In: ICSE, pp. 45–54 (2010)
Sun, C., Lo, D., Khoo, S.C., Jiang, J.: Towards more accurate retrieval of duplicate bug reports. In: ASE, pp. 253–262 (2011)
Tamrawi, A., Nguyen, T., Al-Kofahi, J., Nguyen, T.: Fuzzy set and cache-based approach for bug triaging. In: CSMR, pp. 365–375. ACM, New York (2011)
Thung, F., Lo, D., Jiang, L.: Automatic defect categorization. In: 19th Working Conference on Reverse Engineering (WCRE), 2012, pp. 205–214. IEEE (2012)
Tian, Y., Lawall, J., Lo, D.: Identifying linux bug fixing patches. In: ICSE, pp. 386–396. IEEE (2012a)
Tian, Y., Lo, D., Sun, C.: Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In: WCRE, pp. 215–224 (2012b)
Wang, X., Zhang, L., Xie, T., Anvik, J., Sun, J.: An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information. In: ICSE, pp. 461–470 (2008)
Wu, R., Zhang, H., Kim, S., Cheung, S.C.: Relink: recovering links between bugs and changes. In: SIGSOFT FSE, pp. 15–25 (2011)
Wurst, M.: The word vector tool user guide operator reference developer tutorial (2007)
Xia, X., Lo, D., Wang, X., Yang, X., Li, S., Sun, J.: A comparative study of supervised learning algorithms for re-opened bug prediction. In: CSMR (2013)
Zaman, S., Adams, B., Hassan, A.E.: Security versus performance bugs: a case study on firefox. In: Proceedings of the 8th working conference on mining software repositories, pp. 93–102. ACM, New York (2011)
Zhang, H., Gong, L., Versteeg, S.: Predicting bug-fixing time: an empirical study of commercial software projects. In: Proceedings of the 2013 International Conference on Software Engineering, pp. 1042–1051. IEEE (2013)
Zheng, Z., Wu, X., Srihari, R.: Feature selection for text categorization on imbalanced data. ACM SIGKDD Explor. Newsl. **6**(1), 80–89 (2004)
Zhou, J., Zhang, H., Lo, D.: Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In: ICSE, pp. 14–24. IEEE (2012)
Zimmermann, T., Nagappan, N., Guo, P., Murphy, B.: Characterizing and predicting which bugs get reopened. In: ICSE, pp. 1074–1083 (2012)