The Journal of Systems and Software 165 (2020) 110567

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Bug severity prediction using question-and-answer pairs from Stack Overflow

Youshuai Tan[a], Sijie Xu[a], Zhaowei Wang[b], Tao Zhang[c,*], Zhou Xu[d], Xiapu Luo[e]

[a] College of Software, Harbin Engineering University, Harbin 150001, China
[b] School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China
[c] Faculty of Information Technology, Macau University of Science and Technology, Macao, China
[d] School of Big Data and Software Engineering, Chongqing University, Chongqing, China
[e] Department of Computing, Hong Kong Polytechnic University, Hong Kong, China

ARTICLE INFO

Article history:
Received 29 September 2019
Revised 23 February 2020
Accepted 2 March 2020
Available online 2 March 2020

Keywords:
Stack overflow
Severity prediction
Logistic regression
Bug reports

ABSTRACT

Nowadays, bugs have been common in most software systems. For large-scale software projects, developers usually conduct software maintenance tasks by utilizing software artifacts (*e.g.*, bug reports). The severity of bug reports describes the impact of the bugs and determines how quickly it needs to be fixed. Bug triagers often pay close attention to some features such as severity to determine the importance of bug reports and assign them to the correct developers. However, a large number of bug reports submitted every day increase the workload of developers who have to spend more time on fixing bugs. In this paper, we collect question-and-answer pairs from Stack Overflow and use logical regression to predict the severity of bug reports. In detail, we extract all the posts related to bug repositories from Stack Overflow and combine them with bug reports to obtain enhanced versions of bug reports. We achieve severity prediction on three popular open source projects (*e.g.*, Mozilla, Ecplise, and GCC) with Naïve Bayesian, k-Nearest Neighbor algorithm (KNN), and Long Short-Term Memory (LSTM). The results of our experiments show that our model is more accurate than the previous studies for predicting the severity. Our approach improves by 23.03%, 21.86%, and 20.59% of the average F-measure for Mozilla, Eclipse, and GCC by comparing with the Naïve Bayesian based approach which performs the best among all baseline approaches.

1. Introduction

Software maintenance is a pivotal field in software engineering, due to the increase of the scale and complexity of software projects (Zhang et al., 2015). A lot of software defects have been introduced in the process of development, thus bug fixing has become an important and challenging problem (Xia et al., 2013). To effectively maintain a great number of bugs, many open source software projects (*e.g.*, Mozilla, Bugzilla) use bug tracking systems to keep track of the reported software bugs. Bug tracking systems provide a platform to allow more bugs to be found and resolved so that they can improve the quality of software. A core component in the bug repository are bug reports produced by developers. Bug reports help developers better manage and fix bugs. In the life-cycle of a bug report, the severity, which describes how soon it needs to be fixed is an important attribute in a bug report. We need to confirm the severity level of the bug report to avoid assigning inappropriate developers to perform bug fixing tasks since it may lead to the possible re-assignment. Therefore, it is important to know the severity level of the bug report. Existing studies (Menzies and Marcus, 2008; Tian et al., 2012; Yang et al., 2012b; Zhang et al., 2016; Lamkanfi et al., 2010) have attempted to predict the severity of bug reports. However, the accuracy of these studies is not perfect. The major reason is due to the lack of detailed information in most of the bug reports, which usually have less than 100 words (Zhang et al., 2017).

Stack Overflow (SO) is a website where the questions are diverse and most of them have corresponding answers with high-quality code snippets. Some studies (Yin et al., 2018; Beyer et al., 2018; Reinhardt et al., 2018; Ahasanuzzaman et al., 2018; da Silva et al., 2018; Tahir et al., 2018) use SO as a dataset to improve the accuracy of the experiment. In this paper, we propose a new approach that crawl question-and-answer pairs from SO to enhance the detailed information of bug reports. We give an example to show the pairs have similar information about bug reports in Section 2.2. The logistic regression algorithm

* Corresponding author.
E-mail address: tazhang@must.edu.mo (T. Zhang).

https://doi.org/10.1016/j.jss.2020.110567
0164-1212/

(Vora and Kurzweg, 2016) has a lower cost of computation than other classic machine learning approaches such as Naïve Bayes (Sang-Bum Kim et al., 2006) and Long Short-Term Memory (LSTM) (Graves, 2012). Also, it is appropriate for large-scale data sets than Naïve Bayes (Y. Ng and Jordan, 2002).

To evaluate the effectiveness of our new approach, we conduct experiments on three open source repositories, including Mozilla, Ecplise, and GCC. We choose Naïve Bayesian (Lamkanfi et al., 2010) and KNN (Tian et al., 2012) as baseline approaches. Since a lot of researchers tend to use LSTM to implement the tasks related to bug resolution, we also utilize LSTM to conduct severity prediction to evaluate whether LSTM is still effective for this task. The experimental results show that our proposed method improves average F-measure by 23.03%, 21.86%, and 20.59% for Mozilla, Eclipse, and GCC compared with the Naïve Bayesian (Lamkanfi et al., 2010) which is the best one among our baseline methods.

We summarize the main contributions of our work as follows:

- We crawl question-and-answer pairs from SO to enhance bug reports for improving the accuracy of bug severity prediction.
- We use a lightweight machine learning technology (*i.e.*, logistic regression algorithm) to predict the severity levels of the bug report.
- We conduct experiments on three large open source projects, including GCC, Eclipses, and Mozilla. The results show that the proposed method has better performance than the cutting-edge approaches including Naïve Bayesian (Lamkanfi et al., 2010), and KNN (Tian et al., 2012). Moreover, we also compare it with the approach using LSTM.

The paper is structured as follows. First, Section 2 provides the background knowledge on the bug report, SO, and severity prediction. Our methodology is then proposed in Section 3 followed by the experiment and result revaluation in Section 4 and 5 respectively. Subsequently, we present some threats to validity in Section 6, followed by Section 7 discussing the related work. Finally, Section 8 summarizes the results and introduce the future work.

2. Background knowledge and motivation

In this section, we provide a detailed description of the relevant background information required for our method. First, we introduce some background knowledge about bug reports. Second, we introduce the SO. Third, we discuss the importance of severity prediction and relevant background knowledge.

2.1. Bug report

Software maintenance has become a difficult task in recent years, because fixing bugs costs a lot of project money each year. Software bugs are an inextricable part of software maintenance and development activities, and they may have continuous negative effects such as user inconvenience, inappropriate functionality, and security risk. To better repair the bugs, bug tracking systems such as Bugzilla is used to help the developers manage the bug reports submitted to track errors. A bug report is a tool for tracking the system, which provides a detailed description of the bug report containing all the basic elements, such as pre-defined fields, freeform text, and an attachment. Fig. 1 shows an example of bug reports. The pre-defined fields provide various features of bug reports. The features are determined when the bug report is created, like the report identification number, severity, priority, component, and product.

2.2. Stack overflow

SO is a popular and professional Q&A website that provides a broad knowledge base for software developers. Millions of people ask and answer questions, producing and sharing related programming content. SO's influence increases as more and more developers participant in SO to maintain the quality. Through SO, developers can quickly understand and solve problems. For example, one android developers, posted five design questions on ControlFilter and DrawableContainer class. Since this topic was discussed by users on SO two years ago, it took other android developers just two weeks to find a common solution based on information on SO. Finally, it took very little time for this problem to be resolved from the initial discussion (Ahasanuzzaman et al., 2018).

This is an example of a SO post. In Fig. 2, there is a problem and some answers from different users. Questions can be related to any programming, including programming languages or best practices. In particular, many problems are about how to achieve a specific goal, such as "sorting a list", "merging two dictionaries", or "removing duplicates in lists". In Pengcheng Yin's study (Yin et al., 2018), about 36 percent of the python tags are in the "how-to" category, and they search for these posts which simulate the information of user queries, to obtain parallel data between natural language (NL) and code with fine-grained alignments in SO.

As Fig. 2 shows, the question described the problem briefly and the answer gave a detailed account of the problem and then told the questioner how to solve it. Part of the description of the bug report (id: 518033) is "*I am trying to install Subversive to a freshly installed Eclipse Oxygen RC3 with a new workspace. When I try to install the connectors via Preferences ->Team ->SVN ->SVN Connector I can see that the check for pre-requisites is running, then some dialog pops up that is immediately closed and nothing happens*", which is similar to the answer. We use descriptions of bug reports to realize the automated severity prediction and both questions and answers contain parts which are similar to descriptions of bug reports. Thus we think question-and-answer pairs from SO can help us enhance bug reports.

2.3. Bug severity prediction

A newly submitted bug report needs to undergo three phases, including bug understanding, bug triage, and bug fixing. The three phases and related tasks are in all life-cycle of the bug reports. Bug understanding is the process of verifying bug reports by bug triagers (*i.e.*, fixer). In the process, the status of the bug report is changed from 'Unconfirmed' to 'New'. Bug understanding is to set the stage for the process of bug triage. To better achieve this purpose, bug triager should understand the given bug report, summarize the important content, filter the duplicates, and predict the features (*e.g.*, severity) of the bug report.

Our research mainly focuses on bug severity prediction. Severity is a critical indicator of bug urgency. Severity (*e.g.*, blocker, critical, and major) is the measure to show serious errors and crashes and severity (*e.g.*, normal, minor, and trivial) shows surface errors. Severity is a criterion that can be used in a variety of bugs. For example, severity can be used as an indicator to help determine the time of repair. When dealing with bugs, the software development team decides to repair the current version or delay it to the future version. Such decisions are determined by considering a set of factors, including bugs, their severity, customer stress, and the efforts needed to solve bugs (Snipes et al., 2012). Previous researches use bug reports directly. However, many bug reports lack detailed information (Zhang et al., 2017). This may result in reducing the precision of severity prediction. Thus we use question-and-answer pairs from SO to enhance bug reports.

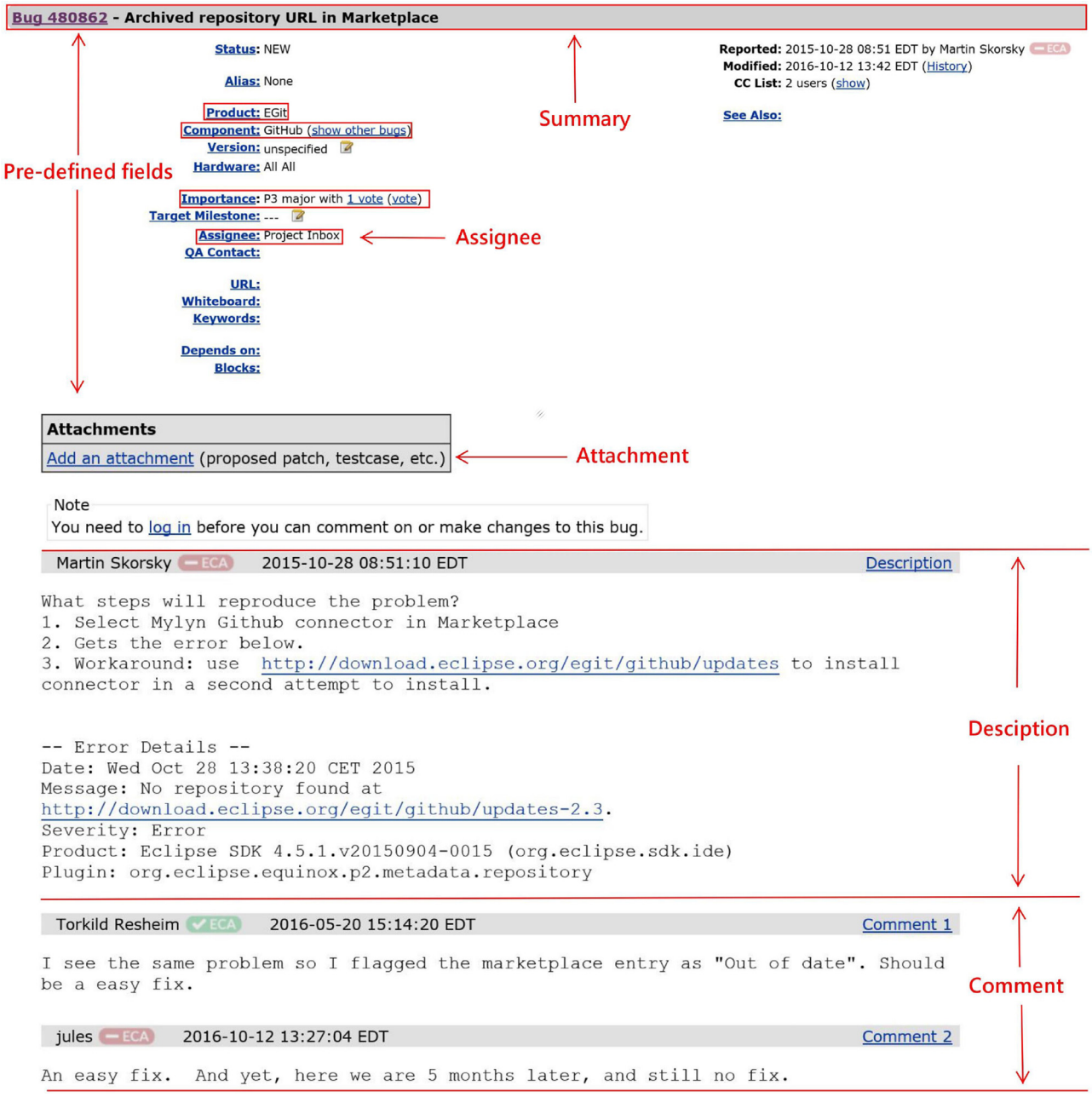

Fig. 1. An example of bug reports.

3. Our methodology for severity prediction

In this section, we describe how to predict the severity level of new bug reports in detail. To show the method clearly, we first give the framework of the proposed method, and then introduce each step of the proposed method in detail.

3.1. Overview

We propose a method based on logistic regression to realize automatic severity prediction. Fig. 3 gives an overview of our method.

The proposed method can be divided into three steps. First, we conduct the pre-processing by mainly using tokenization, stop word removal, and stemming. Second, we download bug reports of the three projects in the past 20 years and crawl question-and-answer pairs of the three projects in SO to enhance bug reports using the logistic regression algorithm. Third, we use the logistic regression classifier to complete the task of severity prediction.

In the following subsection, we describe the three steps in detail.

3.2. Pre-processing

Our work aims to predict the severity of a new coming bug report, which is treated as the task of text classification. Preprocessing is very important for the machine learning algorithm, because it can extract features better and improve the accuracy of the algorithm. We use three kinds of preprocessing: tokenization, stop word removal, and stemming. First of all, the primary task of NLP is to tokenize the text content, that is to say, we divide bug reports into small pieces, so that we can analyze the content of text information more centrally. Furthermore, a set of foreign words (*e.g.*, “to”, “the”, “is”, *etc*) identified in the stop word list are screened to

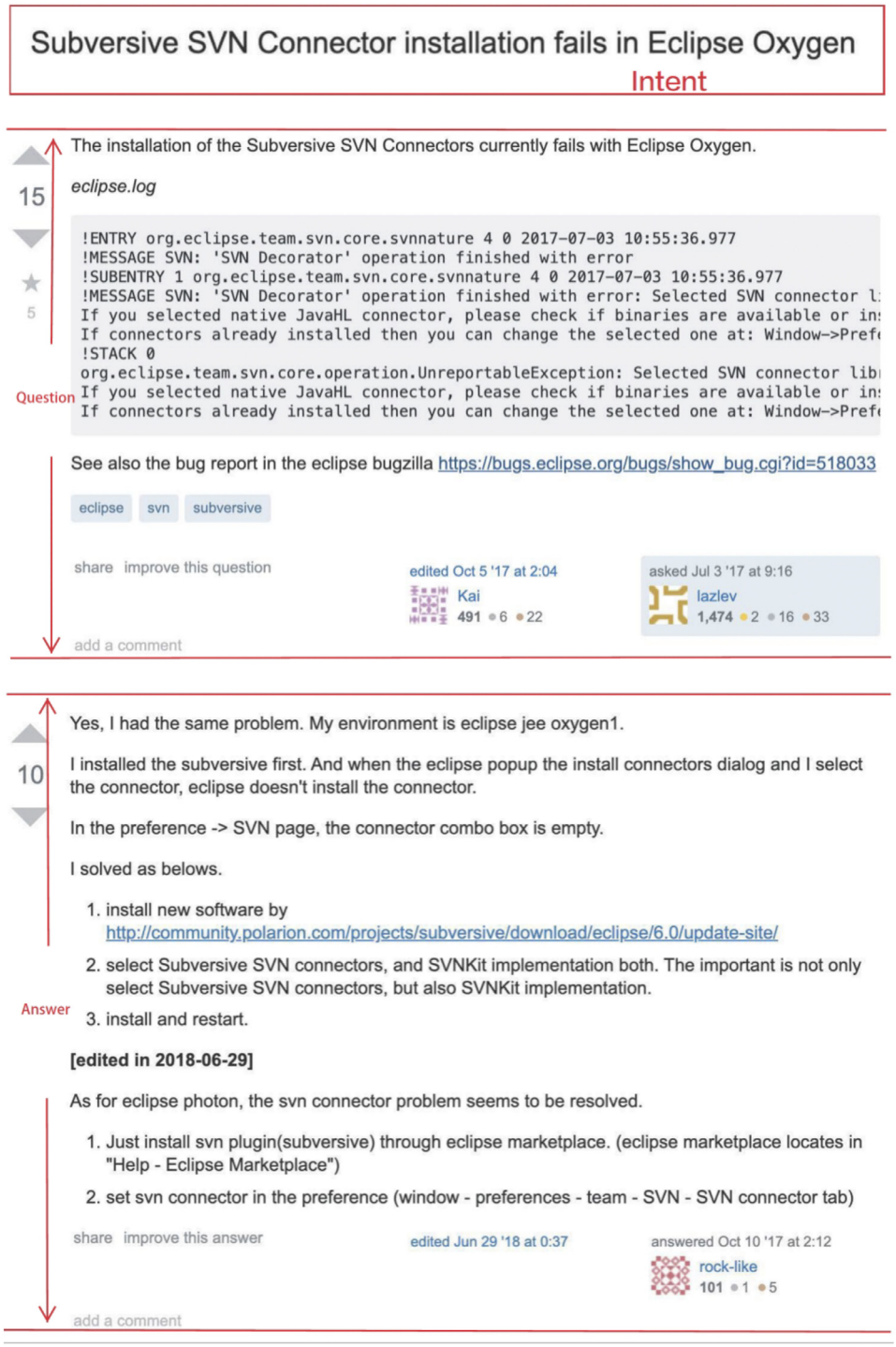

Fig. 2. The example of question-and-answer pair from stack overflow.

ensure the validity of text similarity measurement. Finally, stemming is to extract the stem or root form of words. For example, words such as “drove”, “driving”, and “drives” are changed to “drive”.

In our work, we use a leading platform called NLTK[1] to implement preprocessing activities. This tool provides many useful interfaces with a series of text processing libraries, so it can implement all the required NLP technologies, such as tokenization, stop word removal, and stemming.

3.3. Bug report enhancement by using data from Stack Overflow

There are many question-and-answer pairs about these three projects in SO. Although most question-and-answer pairs do not show any feature of bug reports, their contents may have similar meaning with bug reports. Thus, we crawl the question-and-answer pairs of the projects in SO to enhance the contents of bug

[1] http://www.nltk.org/.

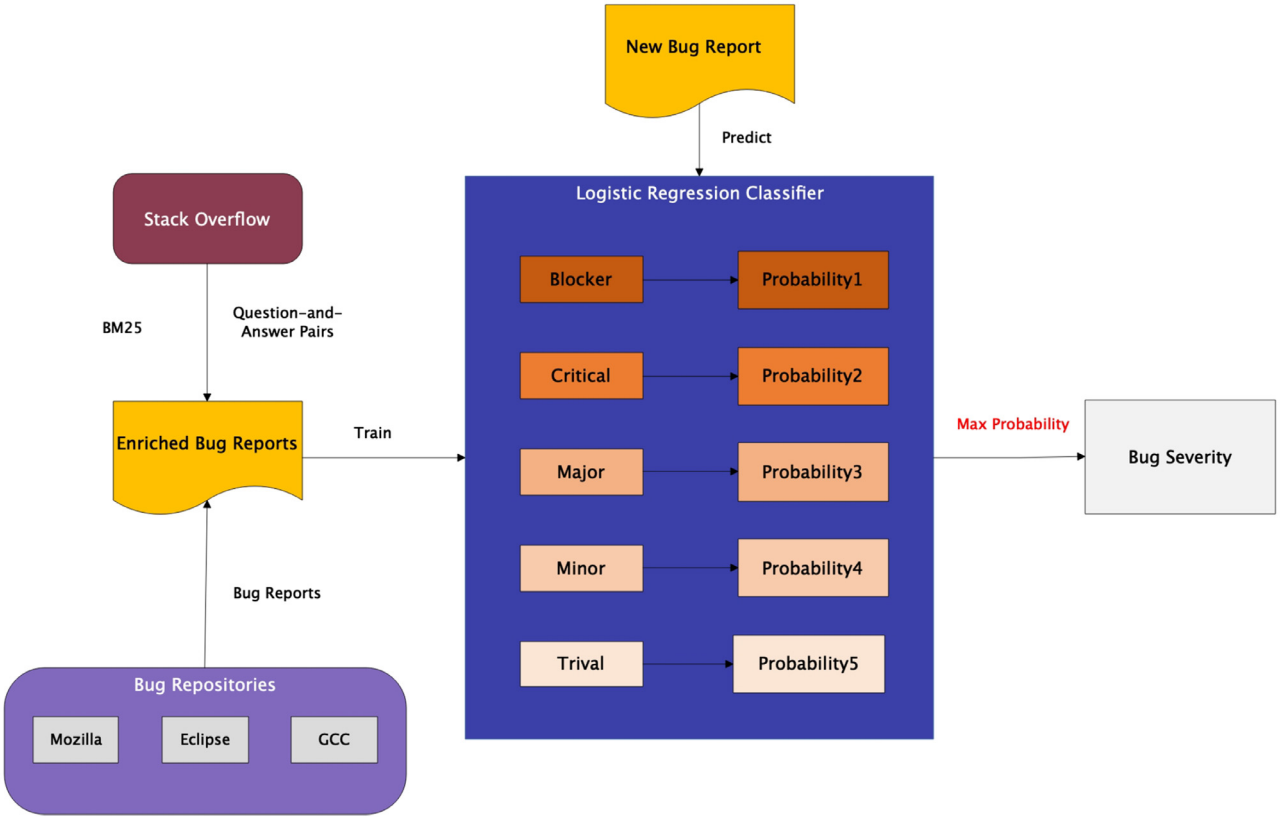

Fig. 3. The overview of our approach.

reports. Each project includes about 500 pairs. We use BM25 to enhance bug reports with sentences in SO.

BM25 is a ranking function developed in the Okapi information retrieval system (Robertson et al., 1994; Robertson and Walker, 1994). In other words, BM25 is a framework consisting of a set of scoring features. We take sentences from SO as query documents QD, $QD = (q_1, q_2, ..., q_n)$. At the same time, we take the bug reports as a corpus CD. As a result, we can get a similarity score between a sentence from SO and a bug report with the formula:

$$BM25(CD, QD) = \sum_{i=1}^{n} idf(q_i) \cdot \frac{f(q_i, CD) \cdot (k_1 + 1)}{f(q_i, CD) + k_1 \cdot (1 - b + b \cdot \frac{|CD|}{avgdl})} \tag{1}$$

The function $f(q_i, CD)$ is the frequency of q_i in CD. $|CD|$ is the length of CD and $avgdl$ is the average length of documents in our corpus. What's more, there are two free parameters k_1 and b. BM25-based scheme has a stable detection performance when k_1 and b are chosen in the commonly used ranges $1.2 \leq k_1 \leq 2.0$ and $0.5 \leq b \leq 0.8$ (Yang et al., 2012a). Thus we set k_1 as 1.5 and set b as 0.75. The function $idf(q_i)$ is the inverse document frequency (IDF) of q_i in our corpus. Its expression is as follows:

$$idf(t) = 1 + \log \frac{C}{1 + df(t)} \tag{2}$$

$idf(t)$ is idf of t, C is the number of documents in our corpus and $df(t)$ is the number of documents that contain t.

As a result, we can use BM25 to get similarity scores between one SO's sentence and each bug report. As we know, the range of scores is uncertain. Thus we divide all the points by the highest score. Then we set nine thresholds 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 and 0.9, after our experiment, we take 0.7 as a threshold. If the similarity is larger than 0.7, we add the sentence to the bug report to enhance our corpus.

3.4. Severity prediction

We use the logistic regression classifier to realize the automated severity prediction. Because the logistic regression algorithm (Vora and Kurzweg, 2016) has a lower cost of computation than other classic machine learning approaches such as Naïve Bayes (Sang-Bum Kim et al., 2006) and Long Short-Term Memory (LSTM) (Graves, 2012). Besides, it is appropriate for large-scale data sets than Naïve Bayes (Y. Ng and Jordan, 2002).

Logistic regression is a classical classification algorithm in the statistical learning method. It can be applied to bi-classification and multi-classification. The multi-nominal logical regression model is as follow:

$$P(Y = k|x) = \frac{exp(\omega_k \cdot x + b_k)}{1 + \sum_{k=1}^{K-1} exp(\omega_k \cdot x) + b_k}, k = 1, 2, ..., K \tag{3}$$

$$P(Y = K|x) = \frac{1}{1 + \sum_{k=1}^{K-1} exp(\omega_k \cdot x) + b_k} \tag{4}$$

In the formulas above, x is a feature vector as the input of the algorithm, and Y is the result of the classification. The domain of Y is the set of all classes 1, 2, 3,..., K. ω_k and b_k are parameters of the algorithm. The value of K is 5, which represents the five labels that we need to predict. We use two different ways to generate feature vectors and compare the results.

First, we use words and POS (Part of speech) Tagging as features. Using words is a straightforward thought. POS Tagging can provide much more semantic information so that it can improve

Table 1
Details of our data set.

Project	# Bug report	# Bug report without the two labels	# Test	Period
Eclipse	77859	7028	1405	10/10/2001-11/05/2019
Mozilla	113357	7164	1432	01/01/2000-15/05/2019
GCC	12812	1189	237	05/03/2000-12/05/2019

the results of classification. We tag all the sentences in bug reports, then combine words and parts of speech as "word-POS". The vocabulary V is composed of the combinations of a word and its parts of speech in our corpus. |V| means the number of combinations in vocabulary V. We turn each bug report into a vector of length |V|. Each dimension represents a combination in V. A value on a dimension represents the number of times the corresponding combination occurs in the bug report. For example, our corpus is "I love bugs.", "I get your love." After removing stop words and POS Tagging, the vocabulary is "love-VB", "bugs-NNS", "get-VB", "love-N". The feature vectors of the two sentences are [1,1,0,0] and [0,0,1,1] respectively.

Second, we use description of bug reports as a corpus to train a Word2Vec (Mikolov et al., 2013) model and get word embedding for every word in our corpus. Word embedding can dig up semantic information as much as it can, which can help to indicate the severity of a bug report. Then we sum the word embeddings of words existing in one bug report and find the parallel unit vector. That unit vector is the feature vector of one bug report.

After conducting experiments shown in Section 5.4, we find that the latter does not perform as accurate as the former does. Thus we take the word and POS Tagging features as training data and use the maximum likelihood method to train the multinominal logical regression model. Then we can get the maximum of likelihood function $\hat{\omega}$. As a result, we can predict the probability of the new bug report belonging to each label and the most probable label is the result. The process is shown in Fig. 3.

4. Experiment setup

In this section, we introduce the experiment setup and propose four research questions to help readers understand our approach.

4.1. Dataset

There are about five large-scale open source projects, including GCC, OpenOffice, Eclipse, NetBeans, and Mozilla. Zhang et al. (2016) found that the distributions of bug reports as per label in NetBeans and OpenOffice present a higher imbalance than the other three data sets, which may affect the reliability of the results, we do not perform the severity prediction in NetBeans and OpenOffice. Thus, we only collect bug reports from the other three systems.

For each system, we only consider five labels for bug reports. We do not consider the reports whose severity label is enhancement because they make no difference to our results. What's more, the "normal" label is the default option for selecting the severity when reporting a bug, therefore, we do not consider them either (Lamkanfi et al., 2010). For them, the experts have labeled the bug reports according to their experience. Tian et al. (2012), Zhang et al. (2016), and Lamkanfi et al. (2010) also believe that the data is reliable.

We then crawl the question-and-answer pairs of SO about the three projects and add them to the bug reports to expand our data set and increase the accuracy of severity prediction. We take 80% of our data as training data and 20% as test data in chronological order. The details of our data sets are described in Table 1.

4.2. Evaluation metrics

To objectively judge the results of the experiment, we use three evaluation indicators in machine learning, Precision, Recall, G_2, and F-measure. Their formulas are as follows:

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

$$G_2 = \sqrt{Recall \times Precision} \tag{7}$$

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recal} \tag{8}$$

In the above formulas, TP (*i.e.*, True Positive instances). represents the number of a test case that is predicted correctly while FP (*i.e.*, False Positive instances) stands for the number of instances predicted falsely in the experiment. FN (*i.e.*, False Negative instances) means the number of actual instances that are not predicted during the period. F-Measure (Goutte and Gaussier, 2005) is the weighted harmonic mean of Precision and Recall, which synthesizes their results. When F-measure is high, it can show that the test method is effective.

4.3. Research questions (RQ)

We mine the question-and-answer pairs from SO to enhance the bug reports data set, then use the logistic regression to achieve severity prediction. We evaluate our approach by answering the following questions:

- **RQ1: How much improvement could our new prediction algorithm gain over state-of-the-art studies such as Naïve Bayesian and KNN?**
  Tian et al. (2012) use KNN for fine-grained bug severity prediction. Lamkanfi and his colleagues were early scholars who proposed to use the machine learning method to realize automated severity prediction (Lamkanfi et al., 2010). It is frequently reported that deep learning tends to perform better than traditional a machine learning in many places. Therefore, it is necessary to compare the above-mentioned studies and deep learning to demonstrate whether our approach performs better than them.
- **RQ2: How to set a threshold based on BM25?**
  The data in SO is very important to our approach, thus we are cautious when we try to add them to bug reports. The accuracy of our method with different thresholds varied and we determine the best threshold through experiments.
- **RQ3: Can question-and-answer pairs from SO help to improve the performance of our algorithm?**
  SO has been famous for its abundant and reliable Q&A pairs. Thus, the information always helps developers to facilitate software maintenance tasks. In this research question, we want to

Table 2
Evaluation results of different approaches using F-measure.

Project	Severity	F-measure (%)			
		Our approach	Naïve Bayesian	KNN	LSTM
Mozilla	Blocker	**100.00**	46.42 (−53.58)	20.47 (−79.53)	10.03 (−89.97)
	Critical	**70.09**	51.21 (−18.88)	45.15 (−24.94)	37.75 (−32.34)
	Major	**47.35**	35.93 (−11.42)	29.80 (−17.55)	34.19 (−13.16)
	Minor	**47.76**	35.60 (−12.16)	25.65 (−22.11)	0.72 (−47.04)
	Trivial	**65.92**	44.64 (−21.28)	7.85 (−58.07)	4.14 (−61.78)
	Average	**66.22**	42.76 (−23.46)	25.78 (−40.44)	17.37 (−48.85)
Eclipse	Blocker	**35.68**	15.28 (−20.4)	4.04 (−31.64)	4.60 (−31.08)
	Critical	**43.50**	30.26 (−13.24)	23.81 (−19.69)	5.15 (−38.35)
	Major	**89.34**	44.74 (−44.60)	46.44 (−42.90)	17.53 (−71.81)
	Minor	**83.93**	56.12 (−27.81)	42.63 (−41.3)	54.33 (−29.60)
	Trivial	**62.79**	16.31 (−46.48)	2.76 (−60.03)	0 (−62.79)
	Average	**63.05**	32.54 (−30.51)	23.93 (−39.12)	16.32 (−46.73)
GCC	Blocker	**40.82**	37.17 (3.65)	7.14 (−33.68)	15.53 (−25.29)
	Critical	**50.00**	53.06 (3.06)	28.83 (−21.17)	17.39 (−32.61)
	Major	**98.28**	36.17 (−62.11)	30.67 (−67.61)	31.58 (−66.70)
	Minor	**57.89**	52.17 (−5.72)	19.78 (−38.11)	20.25 (−37.64)
	Trivial	**73.47**	50.63 (−22.84)	3.64 (−69.83)	32.31 (−41.16)
	Average	**64.20**	45.84 (−18.36)	18.01 (−46.19)	23.41 (−40.79)

know whether Q&A pairs can help to improve the performance of severity prediction by enhancing bug reports.

- **RQ4: Is the accuracy of the severity prediction of our method improved by using Word2Vec?**
  We use one-hot coding to vectorize features. Word2Vec can express a word into vector form quickly and effectively according to the given corpus through the optimized training model, which provides a new tool for applied research in the field of natural language processing. Thus we try this technology to verify whether it improves the accuracy or not.

5. Experiment results

5.1. Answer to RQ1: Comparisons with previous methods

To answer the research question RQ1, we compare our method with the previous frontier research. We choose KNN (Tian et al., 2012), Naïve Bayesian (Lamkanfi et al., 2010), and LSTM as baseline approaches for conducting a comparison. F-measure is a convincing indicator, thus we use it as the standard of comparison for each method. Moreover, we show the experimental results of the comparison in Table 2. The table represents the four approaches' performance for predicting five severity labels in three projects. We give every difference of F-measure compared to our method in parentheses.

For Mozilla, we can predict the blocker, critical, major, minor, and trivial severity labels by F-measure values of 100.00%, 70.09%, 47.35%, 47.76%, and 65.92%. For Eclipse, we can predict the blocker, critical, major, minor, and trivial severity levels by F-measure values of 35.68%, 43.50%, 89.34%, 83.93%, and 62.79%. And for GCC, we can predict the blocker, critical, major, minor, and trivial severity labels by F-measure values of 40.82%, 50.00%, 98.28%, 57.89%, and 73.47% respectively. Our approach outperforms the other three approaches in all projects and labels. However, for the Blocker and critical label of GCC, our approach loses out to Naïve Bayesian by 3.65% and 3.06%.

As a result, we can answer RQ1 as follows:

Answer to RQ1: For severity prediction, our approach outperforms the cutting-edge ones by using enhanced bug reports.

5.2. Answer to RQ2: The setting of threshold

To answer the research question RQ2, we tune the threshold from 0.1 to 0.9 (the step is 0.1). If we set the threshold from 0.1 to 0.6, Data was expanded too much and the F-measure could be nearly 1.0. Thus we only take three thresholds for each of the three items to make polyline maps in Figs. 4–6. The data on the *X*-axis represent the levels of bug severity and the data on the *Y*-axis shows the value of F-measure. For Mozilla, the average F-measure for the three thresholds is 66.22%, 49.86%, and 47.86%. For Eclipse, the average F-measure for the three thresholds is 63.05%, 44.38%, and 39.11%. For GCC, the average F-measure for the three thresholds is 64.20%, 54.44%, and 45.98%. Our approach uses the threshold of '0.7' and we note that the maximum difference is 18.36%, 23.94%, and 18.22%, respectively.

By analyzing the evaluation results using different thresholds, we can answer RQ2 as follows:

Answer to RQ2: We should choose 0.7 as the threshold of BM25 because it can help our approach achieve the best performance.

5.3. Answer to RQ3: The performance of data from Stack Overflow

To answer the research question RQ3, we conduct the experiment to investigate the results of our method without using question-and-answer pairs from SO.

To compare our method with the other method intuitively, we make line charts in Figs. 7–9. For Mozilla, the F-measure values achieve 43.84%, 3.77%, 12.03%, 11.36%, and 15.84% and the Recall values achieve 46.41%, 1.73%, 13.56%, 10.57%, and 9.47% when we predict the severity levels including blocker, critical, major, minor, and trivial. For Eclipse, the F-measure values achieve 17.05%, 14.02%, 38.79%, 28.00%, and 39.26% and the Recall values achieve 17.31%, 10.82%, 36.72%, 31.63%, and 37.46%. And for GCC, the F-measure values achieve 16.51%, 4.03%, 58.31%, 3.54%, and 21.63% and the Recall values achieve 7.49%, 0%, 61.24%, 9.97%, and 27.47%, respectively. Our approach outperforms the other that has no data in SO. We note that the maximum improvement is 61.24%.

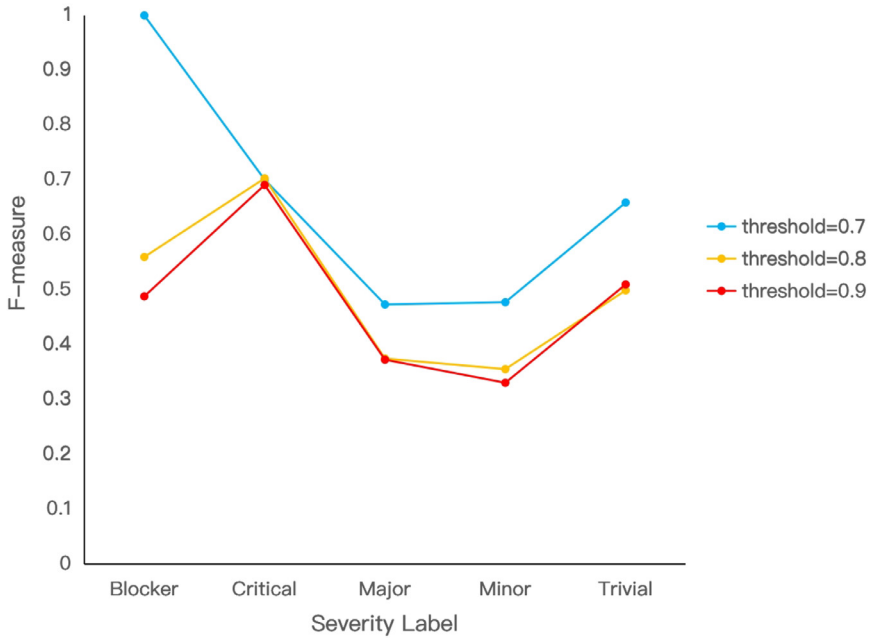

Fig. 4. Performance comparison between different thresholds in Mozilla.

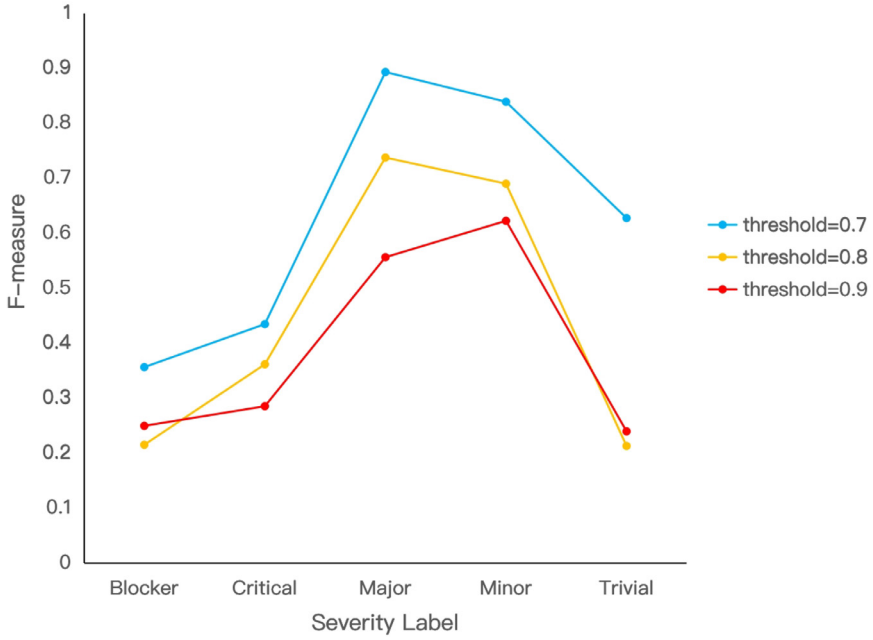

Fig. 5. Performance comparison between different thresholds in Eclipse.

As a result, we can answer RQ3 as follows:

> **Answer to RQ3**: The question-and-answer pairs from the SO make a great contribution to improving the value of Recall and F-measure by enhancing the bug reports.

5.4. Answer to RQ4: The performance of word2Vec

To answer the research question RQ4, we conduct a comparison experiment with the presence of Word2Vec. In this experiment, we use Word2vec to vectorize words and then apply logistic regression to train our classifier. We analyze the results of the comparison in Table 3.

To compare our method with the other method intuitively, we make line charts in Figs. 10–12. For Mozilla, we can predict the blocker, critical, major, minor, and trivial severity labels by F-measure values of 100.00%, 70.09%, 47.35%, 47.76%, and 65.92%. For Eclipse, we can predict the blocker, critical, major, minor, and trivial severity labels by F-measure values of 35.68%, 43.50%, 89.34%, 83.93, and 62.79%. And for GCC, we can predict the blocker, critical, major, minor, and trivial severity labels by F-measure values of 40.82%, 50.00%, 98.28%, 57.89%, and 73.47%, respectively. Our approach outperforms the other that uses Word2Vec. On the three open source projects, we note that the maximum average improvement is 28.28%.

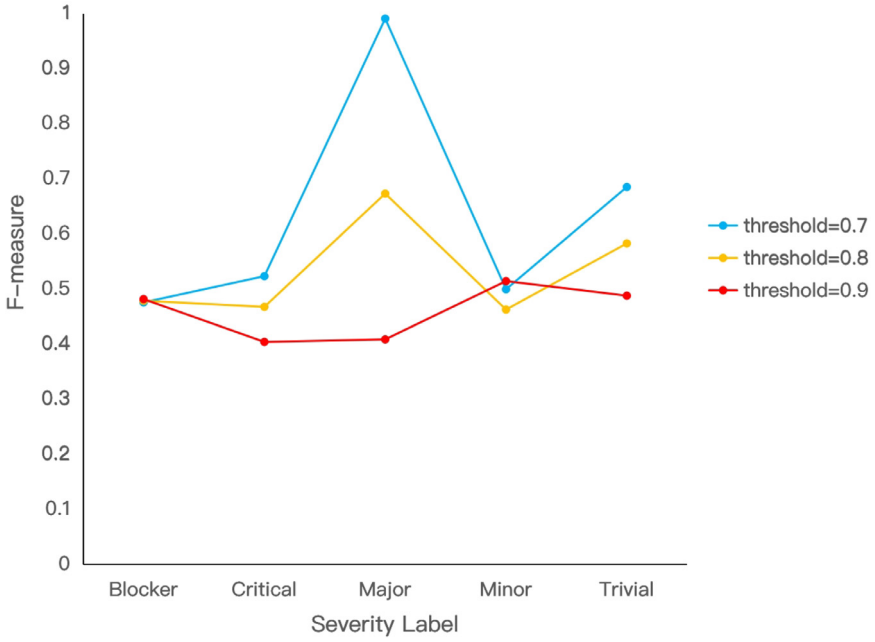

Fig. 6. Performance comparison between different thresholds in GCC.

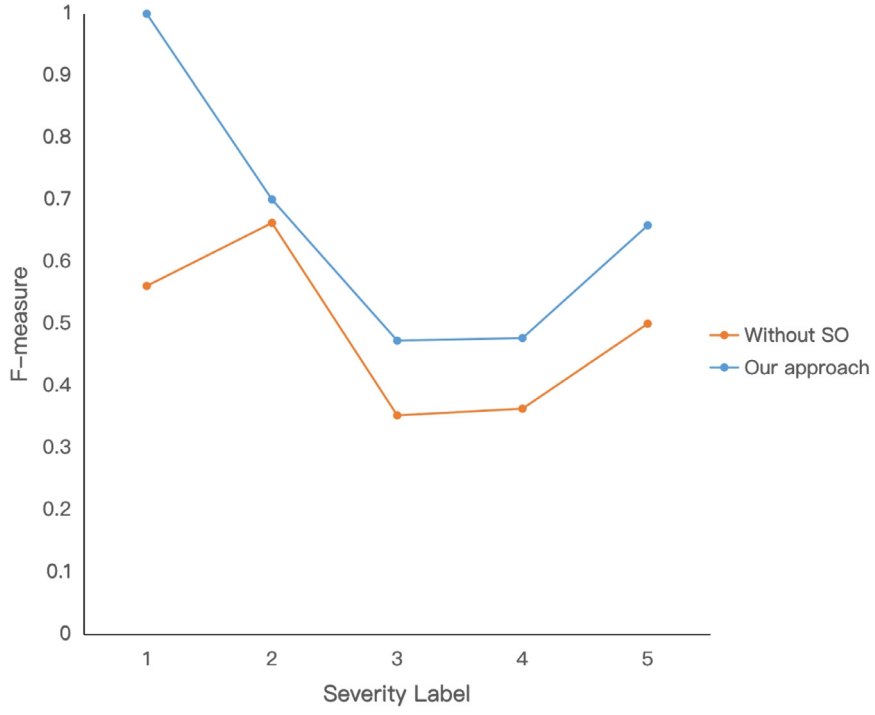

Fig. 7. Performance comparison between using SO data and not in Mozilla

As a result, we can answer RQ4 as follows:

> **Answer to RQ4**: Word2Vec is not suitable for predicting severity. The experiment shows that this reduces the accuracy of the experiment.

5.5. Discussion

In summary, we make good use of question-and-answer pairs from SO to enhance our data. In this way, our data can be more all-round and then the accuracy could be significantly improved. Furthermore, although we crawl many bug reports, only a small part of them whose severity levels are blocker, critical, major, minor, and trivial are used as the training data. Finally, the combination of the two makes our method satisfactory.

6. Threats to validity

In this section, we talk about possible threats to our approach. We divide threats into two categories, external threats, and internal threats. In future work, we will try our best to solve the problems.

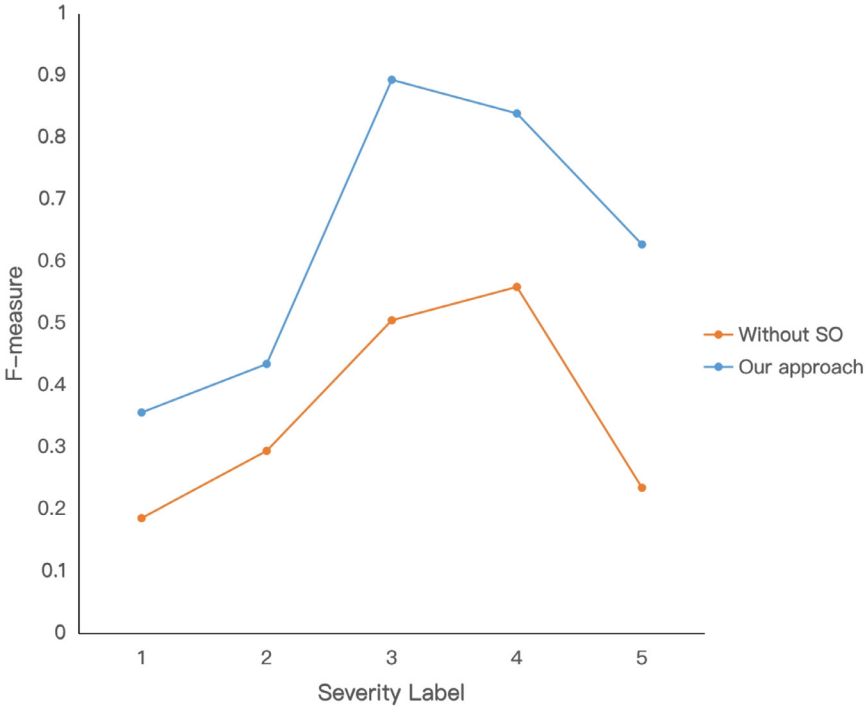

Fig. 8. Performance comparison between using SO data and not in Eclipse.

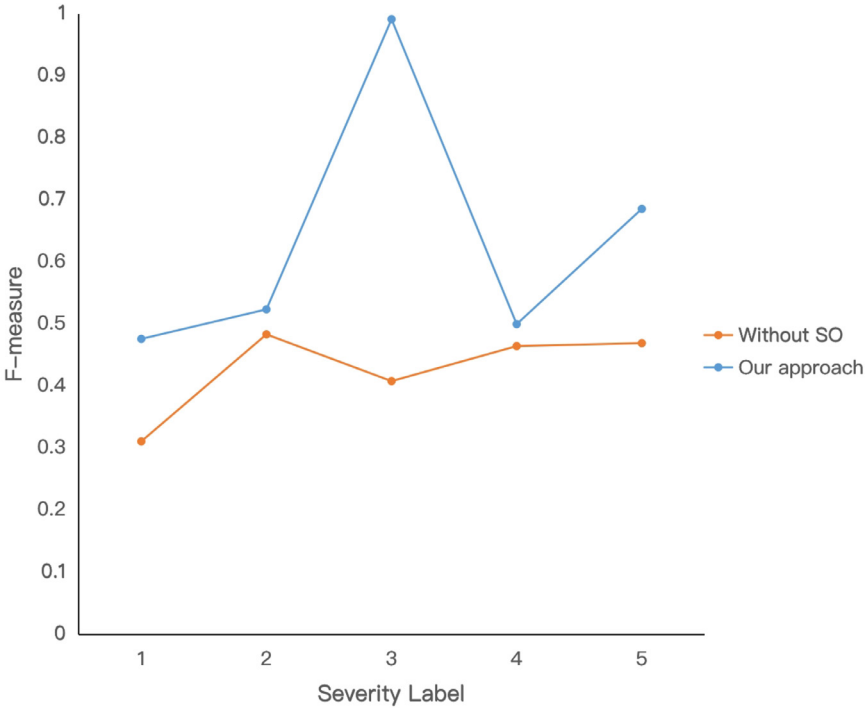

Fig. 9. Performance comparison between using SO data and not in GCC.

6.1. External threats

We have experimented with bug reports extracted from representative large open source projects such as Eclipse and Mozilla, but we are not sure whether the proposed algorithm can be effectively applied to commercial projects. Because the definition of severity in business projects is different from that in open source projects, the current algorithm may not apply to business projects.

6.2. Internal threats

As we known, there is no standard bug report in SO. Thus we can only crawl all the questions and answers related to the projects when we crawl the data. Then these data are added to the bug report database by the BM25 algorithm. Therefore we may have lost some useful statements in the process. What's more, some bug reports of open source projects are of low quality and the labels may not be accurate. Thus the ground truth are not totally correct.

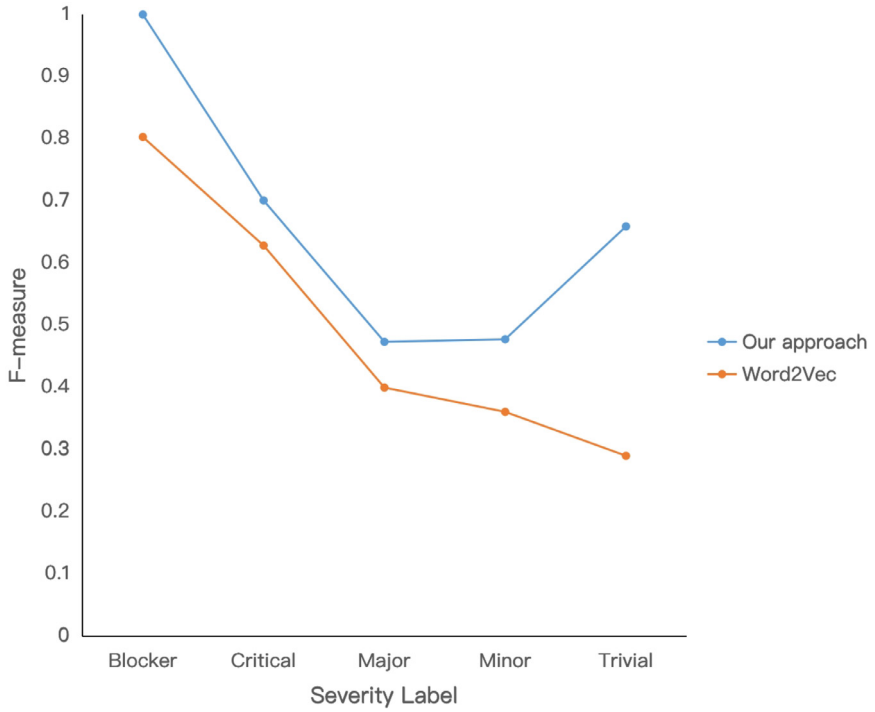

Fig. 10. Performance comparison between our approach using Word2Vec and it removing Word2Vec in Mozilla.

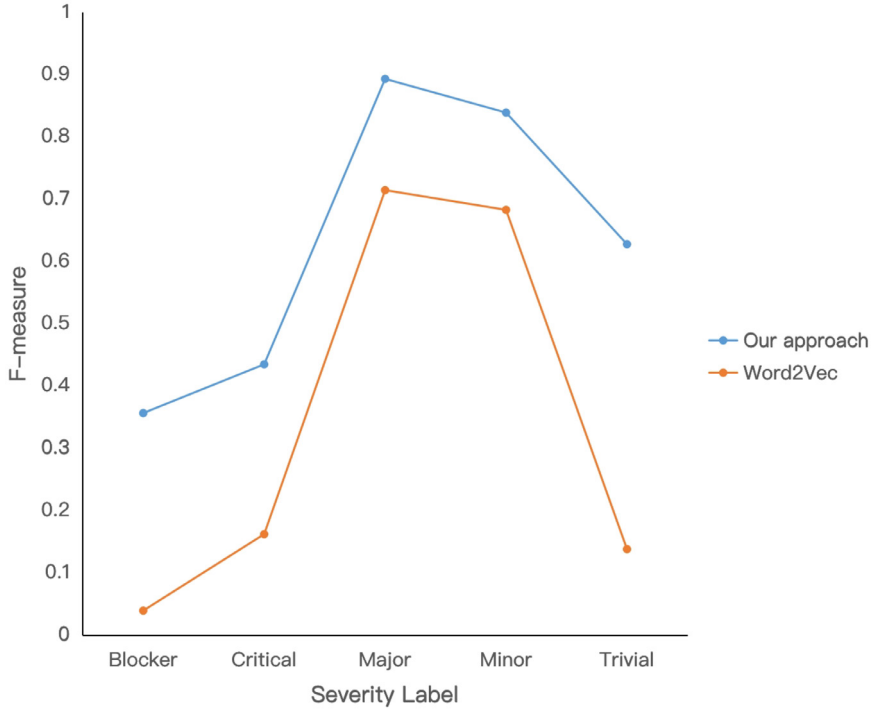

Fig. 11. Performance comparison between our approach using Word2Vec and it removing Word2Vec in Eclipse.

7. Related work

7.1. Stack Overflow

SO has become a basic element of the development toolset, which facilitates system development and helps developers easily and quickly fix bugs. Ponzanelli et al. has developed an Eclipse plugin, called Prompter, which could automatically retrieve the discussion of SO in the given context of IDE (Ponzanelli et al., 2014). Asaduzzaman et al. classified unsolved problems according to SO (Asaduzzaman et al., 2013). Correa and Sureka analyzed and predicted the closed problem in SO (Correa and Sureka, 2013). In another study, they described the problem of being deleted in SO, and set up a prediction model to detect deleted problems at their creation time (Correa and Sureka, 2014).

The researchers often use SO as a data set in the experiment. Aahir et al. used SO for the code odor/antipattern detector because developers used SO to conduct general assessments of code

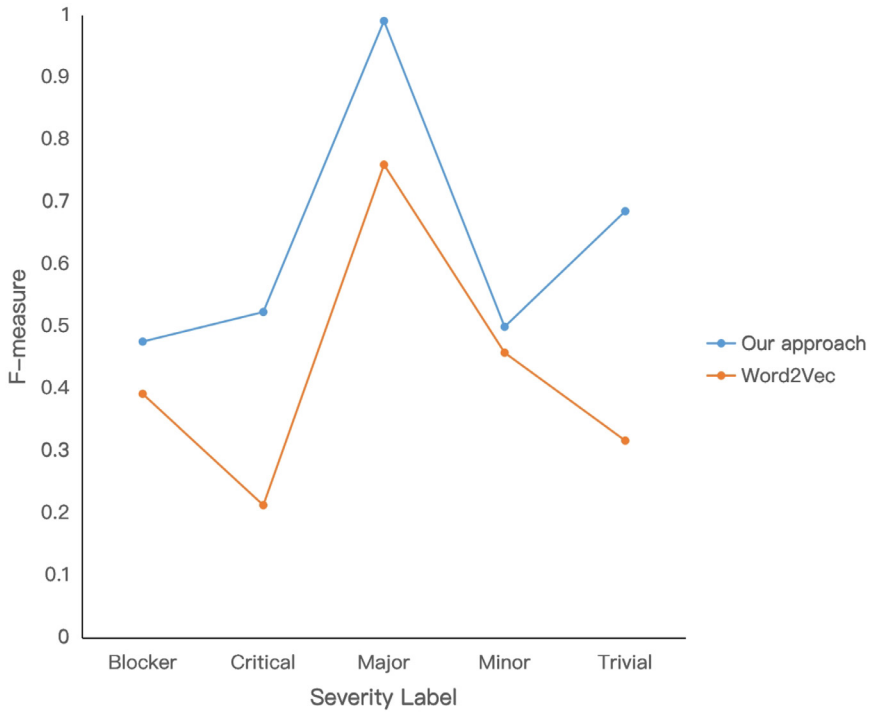

Fig. 12. Performance comparison between our approach using Word2Vec and it removing Word2Vec in GCC.

Table 3
Evaluation results using Word2Vec.

Project	Severity	Using Word2Vec			
		Precision (%)	Recall (%)	G_2 (%)	F-measure (%)
Mozilla	Blocker	69.55	94.98	81.28	80.30
	Critical	56.91	70.16	63.19	62.85
	Major	46.98	34.82	40.45	40.00
	Minor	36.64	35.56	36.10	36.09
	Trivial	35.71	24.44	29.54	29.02
Eclipse	Blocker	25.00	2.15	7.33	3.96
	Critical	42.22	10.05	20.60	16.24
	Major	61.93	84.50	72.34	71.48
	Minor	59.24	80.65	69.12	68.31
	Trivial	68.75	7.69	22.99	13.84
GCC	Blocker	46.51	33.90	39.71	39.22
	Critical	29.63	16.67	22.22	21.33
	Major	61.33	100.00	78.31	76.03
	Minor	44.90	46.81	45.85	45.83
	Trivial	29.55	34.21	31.79	31.71

smells or anti-patterns, rather than demanding specific refactoring solutions (Tahir et al., 2018). Ahasanuzzaman et al. developed a kind of supervised learning method using a Conditional Random Field (CRF) to identify sentences related to API problem in SO (Ahasanuzzaman et al., 2018).

The question-and-answer pairs in SO are written by the professional developers, thus the data is representative which can be used for many tasks. Yin et al. used SO as a data set, because SO can provide a platform where most problems have very high-quality code snippets related to the problems. They obtained parallel data between natural language (NL) and code with fine-grained alignments from SO. The data obtained can be used to complete the code analysis, code retrieval, code summary, and other tasks (Yin et al., 2018).

SO is representative and reliable, thus the types of the problem which are mentioned in SO can represent most types, and developers can carry out statistical classification according to the problems in SO. It is necessary to ensure reliable information in SO, and there are a lot of experimental studies on this (Beyer et al., 2018). Anastasia et al. built a chrome extension to detect online code snippets. They used the API usage pattern from Github to evaluate the API usage on SO posts. The examples of Github which followed common API usage are quantified, and the violations of a given SO fragment are explained (Reinhardt et al., 2018). Silva et al. used the Duppredictor and Dupe to automatically detect the repeated problems in SO preventing its bad effect on the quality (da Silva et al., 2018).

7.2. Severity prediction

The severity of the bug report is an important factor in determining how fast it will take to fix the bug. Therefore, severity prediction attracts lots of developers to study.

Zhou et al. used LDA to explore the severity in each topic, and then studied them on the different platforms, categories and time, and they found that severity could also affect the repair time of bugs (Yang et al., 2012c). Based on previous bug reports, Yang et al. classified bug report topics, and identified the new bug report's topic using multi-feature (et al., component, product, priority, and severity).They recommend the appropriate developer according to the results (Yang et al., 2014). In the early years, Menzies and Marcus proposed a new automated method called SEVERIS to assist engineers in classifying bug reports and helped NASA test engineers identify the severity of each problem correctly (Menzies and Marcus, 2008). Lamkanfi et al. used the text mining technique to determine whether it could accurately predict the severity of a bug report, and the result showed that it could be more accurate in sufficient training set (Lamkanfi et al., 2010). Then they continued to expand the text mining algorithm and compared the four famous text mining algorithms(namely, Naive Bayes, Naive Bayes Multinomial, K-Nearest Neighbour, and Support Vector Machines) to produce the most accurate methods of severe prediction. The results showed that in two open source systems(Eclipse and GNOME) Naïve Bayesian Multinomial performed

better (Lamkanfi et al., 2011). Tian et al. proposed a text similarity function based on BM25, the method using information retrieval could accurately predict bug reports' severity. This fine-grained discriminant method was more accurate than the previous method, such as (SEVERIS (Menzies and Marcus, 2008) Tian et al. (2013). Yang et al. discussed indicators and combined the feature selection schemes (the Multinomial Naïve Bayesian classification approach) based on the Multinomial Naïve Bayesian classification method with severity prediction (Yang et al., 2012b). Zhang et al., used the enhanced version of REP, (*i.e.*, REPtopic) algorithm and KNN classification to collect historical bug reports. They developed a new severity prediction algorithm based on the textual similarities (Zhang et al., 2016). Sahin et al. used the word embedding method to performance feature extraction and used the method of Convolutional Neural Networks (CNN), Long Short Term Memory (LSTM) and Extreme Gradient Boosting (XGBoost) to make severity prediction, and they used severe scores to replace the severity level of the severity prediction (Sahin and Tosun, 2019).

Our work is different from the above-mentioned studies. First, we use question-and-answer pairs to enhance bug reports for improving the performance of severity prediction. Second, we use logical regression to achieve the task. The experimental results show that the enhanced bug reports play an important role in our approach, which can greatly improve the performance. Moreover, the logical regression classifier realizes the accurate automatic severity prediction.

7.3. Bug report management

In the bug understanding, priority is also the object of the developer's regular study, in addition to the severity prediction of the bug report. Priority can help developers verify which bugs should be first noticed. Tian et al. used temporal, textual, author, related-report, severity, and product as features to train a discriminating model and proposed to use machine learning to automatically recommend priority to bug reports (Tian et al., 2015). Umer et al. proposed an automatic priority recommendation method based on the description of emotional words in bug reports (Umer et al., 2018).

Except for severity and priority, the status "reopened" or "blocking" of bug reports often catch developers' eyes because the bug reports which are marked as "reopened" or "blocking" add the bug cost and the fixers' workload. Xia et al. proposed that ELBlocker to determine if a bug is blocking by comparing the boundary value (Xia et al., 2015a).

Duplicates detection is also important to reduce unnecessary time and cost waste. Aggarwal et al. (2015), used context word lists in the SE textbooks and project documentation, and the BM25F similarity method to detect the duplication of the bug report. These software document context lists reflected changes in the process and project of software development. The experimental results showed that the method was the same as the study of Alipour et al. (2013), but the time was less.

There are also a lot of studies on bug triage. Xia et al. proposed DevRec based on bug reporting analysis and developer analysis. DevRec extracted themes, components, products, developers, summaries, and descriptions as recommended features for performing semi-automated fixes on five open source projects. Experimental results showed that DevRec performed better than Wu et al. and Xia et al. (2015b). Xuan et al. used machine learning techniques and reduced the number of data sets to improve the accuracy of bug classification (Xuan et al., 2015). Zhang et al. used REP topic and KNN to find the top K most similar bug reports in a new bug report. The results showed that this method could improve the accuracy of semi-automatic location recommendations.

Many scholars work on bug fixing to reduce workload and repair time. Saha et al. have studied the triage and repair of long-lived bugs, and they extracted bug reports from the bug repository of popular open source projects and analyzed long-lived bugs from five different perspectives: their proportions, severity, distribution, causes, and essential fixes. This study helped developers determine the causes of the delay and improve the entire bug repair process (Saha et al., 2015).

Our approach is related to the above-mentioned studies. Except for software artifacts such as bug reports, source code, commits, we also consider to use SO to enhance the performance of our tasks, which is different from the previous studies.

8. Conclusion

In this paper, we propose a new method to achieve severity prediction. Firstly, we adopt the data in SO, and we extract the posts of the bug repositories to strengthen the bug reports. We make severity predictions about these enhanced bug reports. Then we take words as features, using the maximum likelihood method to train the logistic regression model. Finally, the signature input logic regression model can be obtained, and in this way, the severity of the new bug report can be obtained. To demonstrate our methods, we apply our approach to three popular open bug repositories, including Mozilla, Ecplise, and GCC. The results show that the proposed method transcends the tip method. For severity predictions, our approach is better than the three other methods.

In the future, we will extract more bug reports from commercial software repositories to try our approach in the industry. We also want to use CNN to achieve severity predictions.

Declaration of Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Youshuai Tan: Methodology, Software, Writing - original draft. **Sijie Xu:** Writing - original draft, Validation. **Zhaowei Wang:** Software, Validation. **Tao Zhang:** Conceptualization, Supervision. **Zhou Xu:** Data curation, Writing - review & editing. **Xiapu Luo:** Writing - review & editing, Resources.

Acknowledgements

This work was supported in part by the Natural Science Foundation of Heilongjiang Province (No.LH2019F008), the China Postdoctoral Science Foundation (2017M621247), and the Heilongjiang Postdoctoral Science Foundation (LBH-Z17047).

References

Aggarwal, K., Rutgers, T., Timbers, F., Hindle, A., Greiner, R., Stroulia, E., 2015. Detecting duplicate bug reports with software engineering domain knowledge. In: 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, Montreal, QC, Canada, pp. 211–220.

Ahasanuzzaman, M., Asaduzzaman, M., Roy, C.K., Schneider, K.A., 2018. Classifying stack overflow posts on API issues. In: 25th International Conference on Software Analysis, Evolution and Reengineering, Campobasso, Italy, pp. 244–254.

Alipour, A., Hindle, A., Stroulia, E., 2013. A contextual approach towards more accurate duplicate bug report detection. In: Proceedings of the 10th Working Conference on Mining Software Repositories, San Francisco, CA, USA, pp. 183–192.

Asaduzzaman, M., Mashiyat, A.S., Roy, C.K., Schneider, K.A., 2013. Answering questions about unanswered questions of stack overflow. In: Proceedings of the 10th Working Conference on Mining Software Repositories, San Francisco, CA, USA, pp. 97–100.

Beyer, S., Macho, C., Pinzger, M., Penta, M.D., 2018. Automatically classifying posts into question categories on stack overflow. In: Proceedings of the 26th Conference on Program Comprehension, Gothenburg, Sweden, pp. 211–221.

Correa, D., Sureka, A., 2013. Fit or unfit: analysis and prediction of 'closed questions' on stack overflow. In: Conference on Online Social Networks, Boston, MA, USA, pp. 201–212.
Correa, D., Sureka, A., 2014. Chaff from the wheat: characterization and modeling of deleted questions on stack overflow. In: 23rd International World Wide Web Conference, Seoul, Republic of Korea, pp. 631–642.
Goutte, C., Gaussier, É., 2005. A probabilistic interpretation of precision, recall and *F*-score, with implication for evaluation. In: Advances in Information Retrieval, 27th European Conference on IR Research, Santiago de Compostela, Spain, Proceedings, pp. 345–359.
Graves, A., 2012. Long Short-Term Memory, pp. 37–45.
Lamkanfi, A., Demeyer, S., Giger, E., Goethals, B., 2010. Predicting the severity of a reported bug. In: Proceedings of the 7th International Working Conference on Mining Software Repositories, Cape Town, South Africa, Proceedings, pp. 1–10.
Lamkanfi, A., Demeyer, S., Soetens, Q.D., Verdonck, T., 2011. Comparing mining algorithms for predicting the severity of a reported bug. In: 15th European Conference on Software Maintenance and Reengineering, Oldenburg, Germany, pp. 249–258.
Menzies, T., Marcus, A., 2008. Automated severity assessment of software defect reports. In: 24th IEEE International Conference on Software Maintenance, Beijing, China, pp. 346–355.
Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J., 2013. Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems. Proceedings of a meeting, Lake Tahoe, Nevada, United States., pp. 3111–3119.
Ponzanelli, L., Bavota, G., Penta, M.D., Oliveto, R., Lanza, M., 2014. Mining stackoverflow to turn the IDE into a self-confident programming prompter. In: 11th Working Conference on Mining Software Repositories, Proceedings, Hyderabad, India, pp. 102–111.
Reinhardt, A., Zhang, T., Mathur, M., Kim, M., 2018. Augmenting stack overflow with API usage patterns mined from Github. In: Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Lake Buena Vista, FL, USA, pp. 880–883.
Robertson, S.E., Walker, S., 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In: Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, pp. 232–241.
Robertson, S.E., Walker, S., Jones, S., Hancock-Beaulieu, M., Gatford, M., 1994. Okapi at TREC-3. In: Proceedings of The Third Text REtrieval Conference, pp. 109–126.
Saha, R.K., Khurshid, S., Perry, D.E., 2015. Understanding the triaging and fixing processes of long lived bugs. Inf. Softw. Technol. 65, 114–128.
Sahin, S.E., Tosun, A., 2019. A conceptual replication on predicting the severity of software vulnerabilities. In: Proceedings of the Evaluation and Assessment on Software Engineering, Copenhagen, Denmark, pp. 244–250.
Sang-Bum Kim, Kyoung-Soo Han, Hae-Chang Rim, Sung Hyon Myaeng, 2006. Some effective techniques for naive bayes text classification. IEEE Trans. Knowl. Data Eng. 18 (11), 1457–1466.
da Silva, R.F.G., Paixão, K.V.R., de Almeida Maia, M., 2018. Duplicate question detection in stack overflow: Areproducibility study. PeerJ 6, 981–997. PrePrints
Snipes, W., Robinson, B.P., Guo, Y., Seaman, C.B., 2012. Defining the decision factors for managing defects: a technical debt perspective. In: Proceedings of the Third International Workshop on Managing Technical Debt, Zurich, Switzerland, pp. 54–60.
Tahir, A., Yamashita, A., Licorish, S.A., Dietrich, J., Counsell, S., 2018. Can you tell me if it smells?: a study on how developers discuss code smells and anti-patterns in stack overflow. In: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering, Christchurch, New Zealand, pp. 68–78.
Tian, Y., Lo, D., Sun, C., 2012. Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In: 19th Working Conference on Reverse Engineering, Kingston, ON, Canada, pp. 215–224.
Tian, Y., Lo, D., Sun, C., 2013. DRONE: predicting priority of reported bugs by multi-factor analysis. In: International Conference on Software Maintenance, Eindhoven, The Netherlands, pp. 200–209.
Tian, Y., Lo, D., Xia, X., Sun, C., 2015. Automated prediction of bug report priority using multi-factor analysis. Empir. Softw. Eng. 20 (5), 1354–1383.
Umer, Q., Liu, H., Sultan, Y., 2018. Emotion based automated priority prediction for bug reports. IEEE Access 6, 35743–35752.
Vora, S., Kurzweg, T., 2016. Modified logistic regression algorithm for accurate determination of heart beats from noisy passive RFID tag data. In: IEEE-EMBS International Conference on Biomedical and Health Informatics, pp. 29–32.
Xia, X., Lo, D., Shihab, E., Wang, X., Yang, X., 2015. Elblocker: predicting blocking bugs with ensemble imbalance learning. Inf. Softw. Technol. 61, 93–106.
Xia, X., Lo, D., Wang, X., Zhou, B., 2013. Accurate developer recommendation for bug resolution. In: 20th Working Conference on Reverse Engineering, Koblenz, Germany, pp. 72–81.
Xia, X., Lo, D., Wang, X., Zhou, B., 2015. Dual analysis for recommending developers to resolve bugs. J. Softw. 27 (3), 195–220.
Xuan, J., Jiang, H., Hu, Y., Ren, Z., Zou, W., Luo, Z., Wu, X., 2015. Towards effective bug triage with software data reduction techniques. IEEE Trans. Knowl. Data Eng. 27 (1), 264–280.
Y. Ng, A., Jordan, M., 2002. On discriminative vs. generative classifiers: acomparison of logistic regression and naive bayes. Adv. Neural Inf. Process. Sys 2.
Yang, C., Du, H., Wu, S., Chen, I., 2012. Duplication detection for software bug reports based on BM25 term weighting. In: Conference on Technologies and Applications of Artificial Intelligence, pp. 33–38.
Yang, C., Hou, C., Kao, W., Chen, I., 2012. An empirical study on improving severity prediction of defect reports using feature selection. In: 19th Asia-Pacific Software Engineering Conference, Hong Kong, China, pp. 240–249.
Yang, C., Hou, C., Kao, W., Chen, I., 2012. An empirical study on improving severity prediction of defect reports using feature selection. In: 19th Asia-Pacific Software Engineering Conference, Hong Kong, China, pp. 240–249.
Yang, G., Zhang, T., Lee, B., 2014. Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In: IEEE 38th Annual Computer Software and Applications Conference, Vasteras, Sweden, pp. 97–106.
Yin, P., Deng, B., Chen, E., Vasilescu, B., Neubig, G., 2018. Learning to mine aligned code and natural language pairs from stack overflow. In: Proceedings of the 15th International Conference on Mining Software Repositories, Gothenburg, Sweden, pp. 476–486.
Zhang, T., Chen, J., Jiang, H., Luo, X., Xia, X., 2017. Bug report enrichment with application of automated fixer recommendation. In: 2017 IEEE/ACM 25th International Conference on Program Comprehension, pp. 230–240.
Zhang, T., Chen, J., Yang, G., Lee, B., Luo, X., 2016. Towards more accurate severity prediction and fixer recommendation of software bugs. J. Syst. Softw. 117, 166–184.
Zhang, T., Jiang, H., Luo, X., Chan, A.T.S., 2015. A literature review of research in bug resolution: tasks, challenges and future directions. Comput. J. 59 (5), 741–773.

Youshuai Tan is an undergraduate student in the College of Software at Harbin Engineering University. His research interests are related to Software Engineering and Natural Language Processing.

Sijie Xu is an undergraduate student in the College of Software at Harbin Engineering University. Her research interests are related to Software Engineering and Artificial Intelligence.

Zhaowei Wang is an undergraduate student in the Harbin Institute of Technology. His research area is Natural Language Processing.

Tao Zhang received the BS degree in automation, the MEng degree in software engineering from Northeastern University, China, and the PhD degree in computer science from the University of Seoul, South Korea. After that, he spent one year with the Hong Kong Polytechnic University as a postdoctoral research fellow. Currently, he is an associate professor with the Faculty of Information Technology, Macau University of Science and Technology (MUST). Before joining MUST, he was the faculty member of Harbin Engineering University and Nanjing University of Posts and Telecommunications, China. He published more than 45 high-quality papers at renowned software engineering and security journals and conferences such as the IEEE Transactions on Software Engineering, IEEE Transactions on Information Forensics and Security, IEEE Transactions on Dependable and Secure Computing, IEEE Software, ICSE, etc. His current research interests include mining software repositories and mobile software security. He is a senior member of IEEE and a professional member of ACM.

Zhou Xu is a research assistant professor in the School of Big Data and Software Engineering at Chongqing University, China. He received the Ph.D. degree from Wuhan University, China in 2019. His research interests include software defect prediction, empirical software engineering, feature engineering, and data mining.

Xiapu Luo received the PhD degree in computer science from the Hong Kong Polytechnic University and then spent two years with the Georgia Institute of Technology as a postdoctoral research fellow. He is an associate professor with the Department of Computing, the Hong Kong Polytechnic University. His current research interests include mobile/IoT security and privacy, blockchain, network security and privacy, software engineering, and Internet measurement. He has received seven best paper awards (e.g., INFOCOM18, ISPEC17, ISSRE16, etc.) and one paper received best paper nomination (i.e., ESEM19).