JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 38, 665-680 (2022)
DOI: 10.6688/JISE.202205_38(3).0010

Bug Severity Assessment Based on Weighted Multi-Facet Features with Particle Swarm Optimization*

CHENG-ZEN YANG, KUN-YU CHEN AND ANH-HIEN DAO
Department of Computer Science and Engineering
Yuan Ze University
Chungli, 32003 Taiwan
E-mail: {czyang; kyc12; ahd17}@syslab.cse.yzu.edu.tw

Severity prediction on software bug reports is an important research issue. Recently, many studies have been conducted. Although previous studies have explored different features to facilitate bug severity assessment, the effectiveness of jointly considering these features is not investigated. In the work, multiple features of three facets are collected are studied. Moreover, this study employs a weight adjustment approach using particle swarm optimization (PSO) to find the most appropriate weights of these features. In the prediction framework, three classification models are used to study the influences of these features. The experimental results show that PSO-optimized multi-facet features with the Random Forests model can achieve the best average prediction performance.

Keywords: software bug reports, severity prediction, multi-facet features, weight optimization, particle swarm optimization

1. INTRODUCTION

In software maintenance, severity assessment of bug reports plays an important role because it is influential in the bug fixing process [1,2]. Reporters of bug tracking systems (BTSs), such as Bugzilla [3] and MantisBT [4], can assign the severity levels to the bug reports according to the severity declarations of the BTSs. However, the observations in [5, 6] show that there is a considerable number of bug reports which have been manually assigned with inappropriate severity levels. Therefore, a tool is in high demand to facilitate the verification of the severity assignments of bug reports. Automatic assessment mechanisms are the key issue to predict the severity of a bug report for the severity assessment problem.

In the past decade, various information retrieval approaches and machine learning approaches have been developed for the severity assessment problem, *e.g.*, [1, 5–14]. These approaches are applied to not only mission critical systems [1] but also the open source software projects [5,6]. In these studies, the textual features have been extensively analyzed and exploited because the unstructured textual information provides abundant semantic information related to the severity. To improve the prediction performance, other

Received March 25, 2020; revised June 25, 2020; accepted August 5, 2020.
Communicated by Berlin Chen.
* This work is supported in part by Ministry of Science and Technology, Taiwan under Grants MOST 108-2221-E-155-041, MOST 109-2221-E-155-028, and MOST 110-2221-E-155-052.

features of different facets in bug reports have been also explored. In [10], reporter information is considered in various mining tasks for bug repositories because a reporter with a great community influence may be more intensively involved in processing the severe bugs. In [12], four features related to the quality of bug reports as the *quality indicators* are explored to improve the prediction performance because past studies [15–18] show that several report features, such as *steps to reproduce* and *stack traces*, are important to provide necessary information to developers in fixing bugs. However, these features of different facets have not been jointly studied. In this work, we consider seven features of three facets to investigate their effectiveness: the textual information (T), the quality information (Q), and the reporter information (R). Moreover, the weights of these features are adjusted using the particle swarm optimization (PSO) approach [19] because PSO has shown excellent performance in various optimization problems.

Empirical experiments have been conducted with the Lamkanfi datasets [11] and the collected data from Gentoo Linux. In this work, three classification models, namely Multinomial Naive Bayes (MNB), Support Vector Machines (SVM) [20], and Random Forests (RF) [21], are investigated to study the influences of these features. Four performance measures are used to evaluate the classification models: the area under the curve (AUC) for the Receiver Operating Characteristic curve, precision, recall, and F1 measure. The experimental results show that the performance of these classification models can be benefited from considering these multi-facet features. Moreover, the PSO-based weight adjustment for RF (RF-TQR-PSO) can achieve the best AUC, precision, recall, and F1 measure in all three projects. Compared with the prediction scheme using only textual information, RF-TQR-PSO achieves an improvement of 3.75% in AUC, 3.17% in precision, 0.87% in recall, and 1.45% in F1 measure on average.

The rest of the paper is organized as follows. In Section 2, previous related studies on severity prediction are briefly reviewed. Section 3 explains the proposed prediction scheme with multi-facet features and PSO-based weight adjustment. The studied features are first elaborated. We then describe the prediction scheme with weight adjustment using PSO. Section 4 describes the results of the empirical experiments. Finally, Section 5 concludes the paper.

2. RELATED WORK

For the severity assessment problem, various schemes have been proposed to predict the severity of bug reports [1, 5–14]. Textual information is most commonly used in the previous schemes. In 2008, Menzies and Marcus proposed an automatic severity prediction scheme using text mining and machine learning techniques [1]. Their experimental results show that the proposed scheme performs well for the NASA datasets when the datasets have more than 30 bug reports of high severity levels. However, their scheme cannot have stable performance for different severity levels. Lamkanfi *et al*. discussed four research questions using the Naive Bayes (NB) model [5]. Their study investigated three open source projects, Mozilla, Eclipse, and GNOME. Their experimental results show that the severity semantics can be effectively learned from short summaries rather than long descriptions and some words are the potential *severity indicators*. Thereafter, Lamkanfi *et al*. extended their study to investigate different classification schemes: NB

classifiers, MNB classifiers, 1-Nearest Neighbor (1-NN) classifiers, and SVM [6]. This work shows that the MNB classifiers have the best performance in the AUC measures. In addition, they presented the Eclipse and Mozilla datasets for severity prediction [11]. Considering the importance of severity indicators, Yang *et al*. discussed three feature selection mechanisms to extract the most influential indicators which are given different weights in the MNB model [22]. The experimental results show that properly weighting these indicators can improve the prediction performance.

In 2012, Tian *et al*. proposed an approach calculating the similarity of bug reports to assign the fine-grained severity levels to bug reports [9]. The experimental results show that the approach can get better performance than the previous work of [1]. Recently, Yang *et al*. constructed an emotion dictionary and computed emotion scores between historical and new bug reports [13]. They proposed an MNB-based model called EWD-Multinomial with enhancements of emotion words. Their results show that EWD-Multinomial outperforms the original MNB model. In 2019, Ramay *et al*. used deep learning classification schemes for the severity prediction problem [14]. They proposed a convolutional neural network (CNN) model considering emotion features and textual features. Their experimental results demonstrate the effectiveness of the CNN model.

From the aspect of reporters, Xuan *et al*. studied the severity prediction problem by considering the priorities of developers [10]. The prioritization process is based on the social network technology to find the importance of each reporter for a software product or a component. For severity prediction, the results show that the NB model has improvements only for the Mozilla datasets. In their experiments, the reporter information does not significantly benefit the prediction performance in Eclipse.

From the aspect of the quality of bug reports, Yang *et al*. studied four quality indicators of bug reports to improve the prediction performance [12]. Their experimental results show that using these quality indicators can get performance improvements. However, the influence of feature weighting is not discussed in their study.

3. SEVERITY PREDICTION

In this section, we first describe the problem definitions. Then the studied features of various facets are elaborated. Finally, the PSO-based prediction scheme is described.

3.1 Problem Definition

In this work, we use Bugzilla-based bug reports to study the severity prediction problem. Bugzilla [3] is a well-known BTS used for many open-source projects like Eclipse and Mozilla. It has seven severity levels, namely *blocker*, *critical*, *major*, *normal*, *minor*, *trivial*, and *enhancement*, to manage the bug reports. The severity levels of bug reports are used to show the impact levels of the observed bugs, and they are assigned by reporters. As [6], the severity assessment problem studied in this paper is modeled as a prediction problem to decide the severity level s_i of an incoming report r_x based on the learned experiences.

As indicated in [23], the precise severity assignment in Bugzilla-based bug reports can be an issue because the reporters might lack enough background to assign appropriate severity levels. Despite that this issue is a major concern for precise severity decisions,

Herraiz *et al*. show that correlations exist according to the severity levels of the bug reports [23]. In their work, the bug reports can be generally divided into three groups based on the analysis on the processing time intervals: *Important*, *Non-important*, and *Request for enhancement*. The *Important* group contains bug reports of one of the following levels: *blocker*, *critical*, and *major*. The *Non-important* group is for *normal*, *minor*, and *trivial*. The third group is for *enhancement*, which is not related to any software defect. Similarly, this work considers two severity levels as previous research on severity prediction [5, 6, 12–14]: *Severe* and *Non-Severe*. In this work, the *Severe* class includes the reports assigned to *major*, *critical*, and *blocker*, and the *Non-Severe* class includes the bug reports of the *minor* and *trivial* severity levels. Reports of the *normal* severity are ignored because many of them may be incorrectly assigned [5]. Reports of the *enhancement* severity are also ignored because they are related to functionality strengthening, not software bugs.

3.2 Feature Processing

In this paper, features of three facets in bug reports are jointly considered in this work: the textual information, the quality information, and the reporter information. These features are processed separately according to their properties.

3.2.1 Textual information

In this study, the text of the *summary* field is extracted as the textual information. The textual summary data are processed with the following steps:

- **Tokenization:** In this step, each token is identified and converted to lower case.
- **Stopword removal:** In this step, the stopword tokens are ignored because they do not provide distinctive semantic meanings.
- **Stemming:** The remaining tokens are converted in their stem form. In this work, a Porter stemmer [24] is used for the conversion.

3.2.2 Quality information

As discussed in [15–17], the contents of bug reports contain many important features for developers. In this work, the following four features as the *quality indicators* related to quality of the bug reports are considered.

- **The number of the steps to reproduce:** Bug reports may contain the steps describing how to generate the bugs encountered by the reporters. These steps are usually represented in an enumeration form. Therefore, a parser is designed to find the occurrences of these steps, and the number of the steps is calculated as an indicator.
- **Stack traces:** Stack traces in bug reports generally describe the active stack frames upon some exceptions during the program execution. These traces can help developers find out the origins of the bugs. Since the traces are generated due to exceptions, they tend to appear in severe bug reports. The line number of the traces is calculated as an indicator.

Input: reporter p_i $(1 \leq i \leq n)$ and links among p_i
Output: the importance score RI_i for p_i
1: add a pseudo-root reporter p_0 and the bi-directional links with p_0;
2: initialize $S_0(t=0)=0$ and $S_i(t=0)=1$, $1 \leq i \leq n$;
3: **for** $t=1$ **to** T_c **do** $//T_c$ is the time for convergence
4: compute $S_i(t)=\sum_{j=0}^{n}\frac{W_{ji}}{O_j}S_j(t-1)$
5: **end for**
6: compute $RI_i=\frac{S_i(T_c)+\frac{S_0(T_c)}{n}}{M}$

Fig. 1. Algorithm for calculating the reporter importance scores.

- **The number of attachments:** In a bug report, the reporter may attach additional files to help developers in debugging. These attachments are generally some code segments, documents, or screenshots. A severe bug report usually tends to have some attachments to describe the problem. Therefore, the number of the attachments is considered as an indicator in this work.
- **The report length:** The length of a bug report show how much information it contains. A severe bug report usually tends to have more lines to describe the bug. Our approach considers the line number of each bug report.

These quality indicators are extracted from the long *description* field of bug reports. In this work, all these indicators are normalized between 0 to 1.

3.2.3 Reporter information

As shown in [10], developers have different importance degrees for various collections of bug reports (*i.e.*, bug repositories). For a bug repository, the reporters also form a social network in bug discussions. In this work, the social network model proposed by [10] is used to calculate the reporter importance scores for a dedicated bug repository. The calculation algorithm is shown in Fig. 1, where n is the number of reporters for the bug repository, RI_i is the reporter importance score of the reporter p_i, p_0 is the pseudo-root reporter to connect to all real reporters, $S_i(t)$ is the scoring function as indicated in the algorithm, W_{ji} is the number of comments from reporter p_j to reporter p_i, and O_i is the out-degree of p_i.

$S_i(t)$ is calculated iteratively, and t is used to control the iteration number. T_c is the convergence constant. M is used to normalize the final importance score RI_i and is calculated as follows:

$$M=\frac{S_0(T_c)}{n}+\max_{1\leq i\leq n}S_i(T_c). \tag{1}$$

Thus, the importance score RI_i reflects the impact of reporter p_i on the bug repository. A reporter with a high importance score is usually the person deeply involved in bug reporting. Generally, this person is also involved in the discussions of severe bug reports.

In this work, two kinds of importance scores are calculated according to the scopes of bug repositories: product-based (RI_i^p) and component-based (RI_i^c). These two features

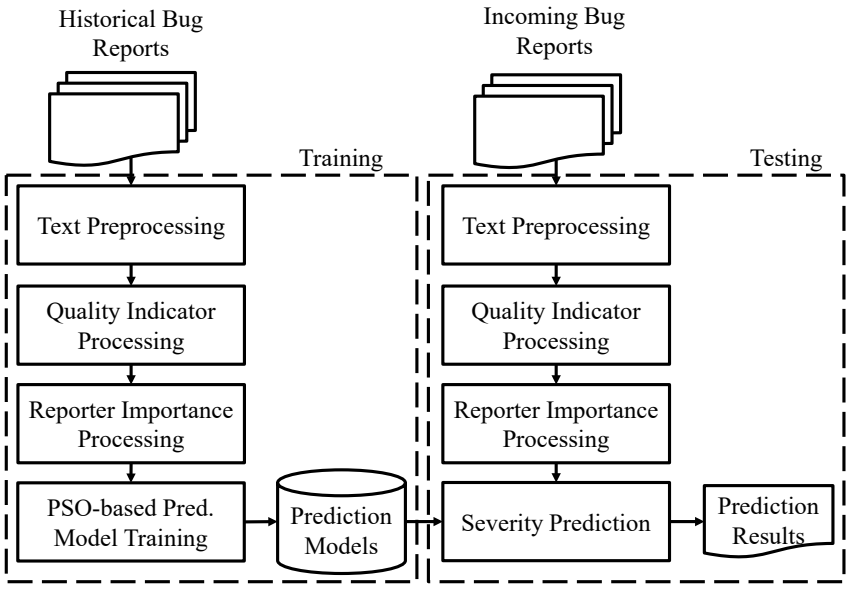


Fig. 2. The PSO-based prediction framework.

represent the different importance degrees of a reporter p_i for a software product and a component.

3.3 Prediction Framework

Fig. 2 illustrates the prediction framework employed in this research. The historical bug reports are first preprocessed using the standard text mining techniques. The aforementioned seven features of three facets are extracted. Based on these features, the classification model is trained using PSO optimization. To investigate the influences of the multi-facet features in severity prediction of bug reports, three classification models are studied in this work: the Multinomial Naive Bayes (MNB) model, Support Vector Machines (SVM) [20], and Random Forests (RF) [21]. In the following, each classification model is briefly presented.

3.3.1 Multinomial naive bayes classification

As shown in [6], the MNB model outperforms other classification models, such as 1-NN and SVM. In this work, MNB is investigated to see its performance changes. In MNB, the probability for a new bug report r_x to be classified into the severity class s_i is defined as $P(s_i|r_x)$, which can be calculated as follows:

$$P(s_i|r_x) = \frac{P(s_i)P(r_x|s_i)}{P(r_x)}. \tag{2}$$

Since $P(r_x)$ is the same for all s_i, $P(s_i|r_x)$ is thus proportional to $P(s_i)P(r_x|s_i)$, which can be expressed as follows:

$$\begin{aligned} P(s_i|r_x) &\propto P(s_i)P(r_x|s_i) \\ &= P(s_i)P(t_1,\ldots,t_n,q_1,\ldots,q_4,ri_p,ri_c|s_i) \\ &= P(s_i)P(t_1|s_i)\times\cdots\times P(t_n|s_i)\times P(q_1|s_i)\times\cdots\times P(q_4|s_i)\times\cdots\times P(ri_c|s_i) \\ &= P(s_i)\times\prod_{1\le l\le n}P(t_l|s_i)\times\prod_{1\le m\le 4}P(q_m|s_i)\times\prod_{o\in\{c,p\}}P(ri_o|s_i). \end{aligned} \tag{3}$$

The priori probability $P(s_i)$ is calculated as the number of training bug reports in s_i divided by the number of total bug reports in the training set. The probability $P(t_l|s_i)$ of token t_l for the severity class s_i is calculated as follows:

$$P(t_l|s_i)=\frac{tf_l}{\sum_{\forall t_v\in V}tf_v}, \tag{4}$$

where tf_l is the term frequency of token t_l and V represents the set of all tokens. To avoid the zero-numerator problem of the MNB model, Laplace smoothing is used for $P(t_l|s_i)$. Thus, $P(t_l|s_i)$ is calculated as follows:

$$P(t_l|s_i)=\frac{1+tf_l}{|V|+\sum_{\forall t_v\in V}tf_v}. \tag{5}$$

3.3.2 Support vector machines

In many classification studies, the Support Vector Machines (SVM) model has shown its impressive performance improvements. For a binary classification problem, SVM tries to find a hyperplane that has the maximum margin between two classes of data $[x_i,y_i]$ such that the optimization problem can be expressed as follows:

$$\min_{w,b,\xi}\ \frac{1}{2}||w||^2+C\sum_{i=1}^{N}\xi_i \quad \text{subject to } y_i(w\varphi(x_i+b))+\xi_i-1,\ \xi\ge 0, \tag{6}$$

where $||w||$ is the norm of the weight vector w in primal space, N is the number of the training data, C is the regulation parameter, $\xi=(\xi_1,\xi_2,\ldots,\xi_N)$ are the slack variables, and b is the bias.

In this work, x_i is a vector that consists of all multi-facet features and $y_i\in$ {severe, non-severe}. In this study, the L2-loss SVM model in LibLinear [25] is employed for severity prediction.

3.3.3 Random forests

In Random Forests (RF), a group of decision trees are constructed using bootstrap aggregating with random feature selection for each split [21]. Through the ensemble process with majority voting, the generalization performance is enhanced.

In RF, there are two parameters controlling the training process: the number of trees and the number of the randomly investigated features. For more trees in RF, the better performance can be obtained as the generalization error converges. However, the training time is increased accordingly. The number of investigated features influences the correlation among trees in RF. As pointed in [21], the performance is promising as the number

Input: number of iterations L and the stopping threshold ε;
Output: the particle x_g with the best weights;
1: initialize each x_i and v_i;
2: initialize the best known position of each particle: $x_i^b = x_i$;
3: **while** iteration $< L$ **and** the stopping difference $> \varepsilon$ **do**
4: **for** $i = 1$ **to** P **do** // P is the number of particles
5: update $v_i = \omega \times v_i + c_1 \times rand() \times (x_i^b - x_i) + c_2 \times rand() \times (x_g - x_i)$;
6: update $x_i = x_i + v_i$;
7: for particle x_i, train the classifier and calculate the fitness function ρ;
8: update the best known position x_i^b for each particle if x_i has a better ρ;
9: update the global best particle x_g if x_i^b has a better ρ;
10: **end for**
11: calculate the stopping difference of ρ values of x_g and the previous x_g;
12: iteration = iteration +1;
13: **end while**

Fig. 3. Framework of the PSO-based weight optimization.

of investigated features is $int(\log_2 M + 1)$, where M is the number of inputs. Although these two parameters can be further tuned with PSO to optimize the performance of RF, they are fixed in this study that aims to investigate the influences of multi-facet features.

3.4 PSO-Based Weight Adjustment

Since there are three facets of information used for severity prediction, this study discusses the performance influences of their weights for the corresponding features. To find the best weights for performance enhancement, Particle Swarm Optimization (PSO) [19] is employed in the training process. In PSO, each particle represents a combination of the seven weights that are characterized its current position x_i and its current velocity v_i. To find the particle x_g of the global best weight combination, PSO iteratively evaluates the particles according to a predefined fitness function. In each iteration, the velocity of each particle are updated as follows:

$$v_i = \omega \times v_i + c_1 \times rand() \times (x_i^b - x_i) + c_2 \times rand() \times (x_g - x_i), \tag{7}$$

and its position is updated as follows:

$$x_i = x_i + v_i, \tag{8}$$

where ω is the inertia weight, c_1 and c_2 are two constants for the learning factors, $rand()$ represents the random number function, and x_i^b is the best known position for x_i.

The PSO optimization algorithm is shown in Fig. 3. In this study, we consider the area under the curve (AUC) values of the Receiver Operating Characteristic (ROC) curves for the fitness function. An ROC curve is defined by the True Positives Rate (TPR) and the False Positives Rate (FPR) as described in the next section.

First, PSO randomly generates an initial group of particles for evolution. In the iteration loop, each particle moves to a new position and the fitness function of each

particle is evaluated to find the global best particle. For the fitness function evaluation, the training dataset is further divided into 10 folds, and we use 10-fold cross-validation to calculate the average AUC result of each particle. Next, the x_i with the best fitness value is selected to update x_i^p. The global best particle x_g is also updated accordingly. This process is iterated until the generation number reaches the termination condition.

4. EXPERIMENTS

4.1 Experiment Setup

In this work, two research questions are discussed to study the influences of these multi-facet features:

1. Can the joint consideration of features of various facets benefit the severity prediction performance?
2. Can the proposed PSO-based weight adjustment approach effectively improve the severity prediction performance?

To answer the research questions and evaluate the effectiveness of the proposed PSO-based approach, we have conducted empirical experiments with the datasets of Eclipse and Mozilla collected by Lamkanfi *et al*. [11] and the datasets of Gentoo Linux. Table 1 shows the details of these datasets in which SBR represents the severe bug reports and NSBR represents the non-severe bug reports. Table 1 also shows the abbreviations of the datasets. For example, EPU represents the dataset extracted from the component *UI* of the product *Platform* in the *Eclipse* project.

Table 1. The details of the experimental datasets.

Project	Product:Component	# SBR	# NSBR	Period
Eclipse	Platform:UI (EPU)	1,004	428	2006/1-2011/3
Eclipse	JDT:UI (EJU)	336	482	2006/1-2011/3
Mozilla	Core:Layout (MCL)	949	184	2006/1-2013/12
Mozilla	Firefox:General (MFG)	10,227	2,430	2006/1-2013/12
Gentoo	Linux:Core System (GLC)	1,302	608	2006/1-2013/12
Gentoo	Security:Vulnerabilities (GSV)	686	2,001	2006/1-2013/12

Table 2 shows the confusion matrix for binary severity prediction evaluation. TP represents the number of correctly predicted severe reports. FN represents the number of wrongly predicted non-severe reports. FP represents the number of wrongly predicted severe reports. TN represents the number of correctly predicted non-severe reports.

Table 2. Confusion matrix for the severity prediction problem.

Actual class	Predicted Severe	Predicted Non-severe
Actually Severe (Positive)	TP	FN
Actually Non-severe (Negative)	FP	TN

As previous studies [5,6], this work uses the area under the curve (AUC) values of the Receiver Operating Characteristic (ROC) curves as the main model evaluation measure. Therefore, the AUC measure is also used for the evaluation of the fitness function in PSO. Moreover, traditional precision, recall, and F1 measure are also discussed for analysis. They are calculated as follows:

$$TPR = TP/(TP+FN), \tag{9}$$

$$FPR = FP/(FP+TN), \tag{10}$$

$$Precesion = TP/(TP+FP), \tag{11}$$

$$Recall = TP/(TP+FN), \tag{12}$$

$$F1 = 2 \times Precision \times Recall/(Precision+Recall). \tag{13}$$

In the experiments, the performance is measured using a 10-fold cross-validation approach. Each dataset is first equally divided into 10 parts and each part does not contain any bug report of other parts. Then 9 parts are used to train the classifier. The remaining part is used for testing. The testing process is repeated 10 times and 10 parts are all tested in a rotational manner. For PSO settings, 40 particles are used, the iteration control is 500, the stopping threshold ε is 0.0001, the inertia weight ω is 0.9, and two acceleration constants c_1 and c_2 are 2. In this study, the classification models are trained in WEKA [26]. For SVM, we use the L2-loss SVM model in LibLinear [25], and the tolerance of the termination criterion is 0.001. For Random Forests, the number of trees is 50 with other default settings.

The significance of the quality information to the severity is further statistically analyzed using Pearson Chi-square univariate tests for steps to reproduce, stack traces, and attachments. For the report length, we calculate the average line number per bug report for six datasets. In Chi-square univariate tests, each independent variable is considered as a binary value for each bug report. The null hypothesis (H_0) is that there is no significant difference between the investigated variable and the observed outcomes, namely the *Severe* decisions. If $p \leq 0.05$, the null hypothesis is rejected. Table 3 presents the statistical results, which show that the occurrences of steps to reproduce and attachments are significant to the *Severe* decision in most datasets. For stack traces, the null hypothesis is rejected only in the Eclipse datasets EPU and EJU. The p-values are all 1.000 in MCL, GLC, and GSV because the bug reports of these datasets do not have any stack trace.

Table 3. The p-values of Chi-square analysis.

	EPU	EJU	MCL	MFG	GLC	GSV
Steps to reproduce	0.004	0.006	0.086	0.000	0.003	0.005
Stack traces	0.000	0.000	1.000	0.911	1.000	1.000
Attachments	0.001	0.000	0.746	0.000	0.000	0.000

Table 4 shows the average line numbers in the *Severe* and *Non-Severe* classes of six datasets. The results show that a severe bug report is longer than a non-severe bug report on average.

Table 4. The average line numbers of the datasets.

	EPU	EJU	MCL	MFG	GLC	GSV
Severe	26.06	34.57	26.88	24.36	45.91	18.34
Non-Severe	10.72	13.33	15.45	21.29	30.15	14.76

4.2 Experimental Results

In the experiments, four feature configurations are discussed according to their facets. The configuration T represents that only the textual information (T) is considered in the severity prediction. In the configuration TQ, the textual information and the quality indicators (Q) of bug reports are used for classifier training and testing. The configuration TR represent that the textual information and reporter importance scores (R) are used. In the configuration TQR, the textual information, the quality indicators, and the reporter importance scores are all considered. For example, MNB-TQR represents that the MNB model is used, and features of three facets T, Q, and R are all considered.

4.2.1 RQ1: Influences of features

Since RQ1 concerns the effectiveness of different joint considerations of features, all features are equally considered for classification model training, *i.e.*, all weights of features are set to 1. Table 5 shows the average performance of all classification models on six datasets. The best results for the performance measures are highlighted in bold face. As shown in Table 5, considering more features can improve the performance in most cases. Among all classification configurations, RF-TQR outperforms other configurations in AUC, precision, and F1 measure on average. The experimental results show that the joint consideration of multi-facet features can generally improve the prediction performance. The results also show that RF generally has the best performance among three classification models.

Since RF can achieve the best performance, its prediction performance is further discussed for each dataset. Table 6 lists the performance results of RF in six datasets. For AUC performance, considering only textual information, namely RF-T, has the lowest performance in all datasets. Generally RF-TQR can achieve the best AUC performance

Table 5. Average performance of six datasets for all classification models.

Config.	AUC	Precision	Recall	F1
MNB-T	0.8065	0.7727	0.8022	0.7870
MNB-TQ	0.8112	0.7745	0.8127	0.7930
MNB-TR	0.8080	0.7745	0.7990	0.7863
MNB-TQR	0.8128	0.7775	0.8072	0.7918
SVM-T	0.6838	0.7663	0.7948	0.7800
SVM-TQ	0.6917	0.7755	0.7973	0.7857
SVM-TR	0.6872	0.7680	0.7997	0.7832
SVM-TQR	0.6938	0.7772	0.8007	0.7883
RF-T	0.8085	0.7612	0.8410	0.7947
RF-TQ	0.8240	0.7703	0.8390	0.7958
RF-TR	0.8190	0.7643	**0.8442**	0.7968
RF-TQR	**0.8328**	**0.7822**	0.8428	**0.8025**

Table 6. Performance results of different RF configurations in each dataset.

Dataset	Config.	AUC	Precision	Recall	F1
EPU	RF-T	0.766	0.764	0.953	0.848
	RF-TQ	0.791	**0.764**	0.953	0.848
	RF-TR	0.780	0.753	0.958	0.843
	RF-TQR	**0.811**	0.757	**0.964**	**0.848**
EJU	RF-T	0.777	0.658	**0.595**	0.625
	RF-TQ	0.814	0.708	0.577	0.636
	RF-TR	0.779	0.670	0.580	0.622
	RF-TQR	**0.824**	**0.736**	0.589	**0.655**
MCL	RF-T	0.910	0.872	0.982	0.924
	RF-TQ	0.919	0.872	**0.997**	**0.930**
	RF-TR	0.917	0.871	0.989	0.926
	RF-TQR	**0.922**	**0.874**	0.991	0.929
MFG	RF-T	0.776	0.828	0.973	0.895
	RF-TQ	0.784	0.825	0.985	0.898
	RF-TR	0.786	**0.829**	0.981	0.899
	RF-TQR	**0.788**	0.824	**0.989**	**0.899**
GLC	RF-T	0.763	0.736	0.944	0.827
	RF-TQ	0.764	0.736	0.958	0.832
	RF-TR	**0.777**	**0.742**	0.955	0.835
	RF-TQR	0.767	0.740	**0.963**	**0.837**
GSV	RF-T	0.859	0.709	0.599	0.649
	RF-TQ	0.872	0.717	0.564	0.631
	RF-TR	0.875	0.721	**0.602**	**0.656**
	RF-TQR	**0.885**	**0.762**	0.561	0.647

for different software projects.

For the precision and recall performance, the influences of the quality information and the reporter information vary. In most cases, the quality information and the reporter information can have benefits to improve the prediction performance.

Regarding to the software projects, the results show that the improvements of the joint consideration of multi-facet features in Mozilla are comparatively minor. In our manual investigation on the Mozilla datasets, we find that the amount of quality information is relatively small in comparison with the textual information. Moreover, the reporter social networks in Mozilla are more imbalanced because there are numerous reporters but only a small number of reporters is dominant in the social network. Therefore, the advantage of the quality information and the reporter information cannot be completely utilized. This situation is more obvious for the MNB and SVM models. The performance measures of four configurations of MNB are very close in Mozilla, and four SVM configurations also have close performance measures.

4.2.2 RQ2: PSO-based weight adjustment

For RQ2, the weights of the seven kinds of features are adjusted using PSO. The training data is further divided into the PSO-training set and the PSO-validation set using the 10-fold cross-validation approach to find the best weight configurations. The adjusted weights are applied to the testing data for performance evaluation. The investigations

on the distributions of these weights show that the distributions are varied because PSO dynamically finds the best weights according to the characteristics of the training data.

Table 7 shows the performance results of the PSO-based weight adjustment. The experimental results show that all PSO-based classification models, namely MNB-TQR-PSO, SVM-TQR-PSO, and RF-TQR-PSO, can take the advantage of weight adjustment in the average AUC and precision measures.

Table 7. Performance comparison of weight adjustment.

Config.	AUC	Precision	Recall	F1
MNB-T	0.8065	0.7727	0.8022	0.7870
MNB-TQR	0.8128	0.7775	0.8072	0.7918
MNB-TQR-PSO	0.8320	0.8005	0.7712	0.7798
SVM-T	0.6838	0.7663	0.7948	0.7800
SVM-TQR	0.6938	0.7772	0.8007	0.7883
SVM-TQR-PSO	0.6990	0.7818	0.7772	0.7778
RF-T	0.8085	0.7612	0.8410	0.7947
RF-TQR	0.8328	0.7822	0.8428	0.8025
RF-TQR-PSO	**0.8388**	**0.7853**	**0.8483**	**0.8062**

In AUC, MNB-TQR-PSO achieves an improvement of 3.16% over MNB-T on average, SVM-TQR-PSO achieves an improvement of 2.22% over SVM-T on average, and RF-TQR-PSO achieves an improvement of 3.75% over RF-T on average. These results show that considering the AUC performance in PSO effectively improves the performance of all studied classification models for AUC.

Similarly, MNB-TQR-PSO on average achieves an improvement of 3.60% in precision over MNB-T, SVM-TQR-PSO on average achieves an improvement of 2.02% over SVM-T, and RF-TQR-PSO on average achieves an improvement of 3.17% over RF-T. These results show that the number of the data predicted false-positive can be effectively reduced in all three classification models.

In recall, MNB-TQR-PSO and SVM-TQR-PSO have the bottom average recall and F1 performance in comparison to other MNT-based and SVM-based approaches, respectively. The results show that the PSO optimization process hinders the effectiveness of identifying severe bug reports for the MNB and SVM models. However, the RF model can still achieve the best recall (0.8483) and F1 (0.8062) performance on average. Since the RF model is an ensemble approach that has many decision-tree classification models trained on a random subset of the features, it can mitigate the over-fitting problem potentially introduced in the PSO optimization process.

5. CONCLUSIONS

Severity of bug reports is important for bug fixing in software maintenance. In the past, many studies have been conducted for this issue. In this paper, we propose a PSO-based prediction scheme based on multi-facet information of bug reports to improve the prediction performance. In this work, three classification models, namely MNB, SVM, and RF, are investigated to study the benefits of using these features. Since these features have different influences on prediction performance, PSO is employed to adjust their

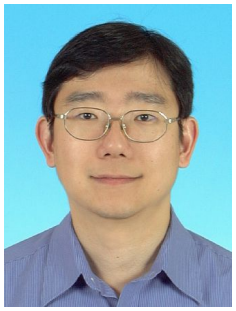
weights for performance improvements. We have conducted empirical experiments with six software components from three open source projects: Eclipse, Mozilla, and Gentoo Linux. The experiments investigate the effectiveness of multi-facet information in severity prediction. The results show that exploiting the multi-facet information can improve the performance of severity prediction. We also investigate the impacts of weight adjustment with PSO. The results show that the optimized weights can further improve the average performance of AUC and precision of all three classification models for all datasets. In this study, the RF-TQR-PSO can achieve the best average performance on AUC, precision, recall, and F1 measure.

In our future study, some issues will be investigated. First, we plan to explore other information of bug reports, such as the historical fixing file locations. The file locations may provide a different insight of the bug severity. Second, the classification model can be further improved to obtain more accurate prediction performance. Since the RF model achieves the outstanding performance in the current study, the ensemble approach shows its research potential in the future. The emerging deep learning technology will be also explored for its superior performance in other related tasks of software repository mining.

REFERENCES

1. T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Proceedings of the 24th IEEE International Conference on Software Maintenance*, 2008, pp. 346-355.
2. M. Sharma, M. Kumari, and V. B. Singh, "Multi-attribute dependent bug severity and fix time prediction modeling," *International Journal of System Assurance Engineering and Management*, Vol. 10, 2019, pp. 1-25.
3. Bugzilla, "Mozilla bug tracking system," https://bugzilla.mozilla.org/.
4. MantisBT, "Mantis bug tracker," https://www.mantisbt.org/.
5. A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories*, 2010, pp. 1-10.
6. A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug," in *Proceedings of the 15th European Conference on Software Maintenance and Reengineering*, 2011, pp. 249-258.
7. K. K. Chaturvedi and V. B. Singh, "Determining bug severity using machine learning techniques," in *Proceedings of the 6th International Conference on Software Engineering*, 2012, pp. 1-6.
8. R. Gangadharaiah and R. Catherine, "PriSM: Discovering and prioritizing severe technical Issues from product discussion forums," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, 2012, pp. 1627-1631.
9. Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *Proceedings of the 19th Working Conference on Reverse Engineering*, 2012, pp. 215-224.

10. J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *Proceedings of the 34th International Conference on Software Engineering*, 2012, pp. 25-35.
11. A. Lamkanfi, J. Pérez, and S. Demeyer, "The Eclipse and Mozilla defect tracking dataset: A genuine dataset for mining bug information," in *Proceedings of the 10th IEEE Working Conference on Mining Software Repositories*, 2013, pp. 203-206.
12. C.-Z. Yang, K.-Y. Chen, W.-C. Kao, and C.-C. Yang, "Improving severity prediction on software bug reports using quality indicators," in *Proceedings of the 5th IEEE International Conference on Software Engineering and Service Science*, 2014, pp. 216-219.
13. G. Yang, S. Baek, J.-W. Lee, and B. Lee, "Analyzing emotion words to predict severity of software bugs: A case study of open source projects," in *Proceedings of the 32nd ACM SIGAPP Symposium on Applied Computing*, 2017, pp. 1280-1287.
14. W. Y. Ramay, Q. Umer, X.-C. Yin, C. Zhu, and I. Illahi, "Deep neural network-based severity prediction of bug reports," *IEEE Access*, Vol. 7, 2019, pp. 46 846-46 857.
15. N. Bettenburg, S. Just, A. Schröter, C. Weiß, R. Premraj, and T. Zimmermann, "Quality of bug reports in Eclipse," in *Proceedings of OOPSLA Workshop on Eclipse Technology eXchange*, 2007, pp. 21-25.
16. N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Extracting structural information from bug reports," in *Proceedings of the 5th International Working Conference on Mining Software Repositories*, 2008, pp. 27-30.
17. A. Schröter, N. Bettenburg, and R. Premraj, "Do stack traces help developers fix bugs?" in *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories*, 2010, pp. 118-121.
18. T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, and C. Weiss, "What makes a good bug report?" *IEEE Transactions on Software Engineering*, Vol. 36, 2010, pp. 618-643.
19. J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*, Vol. 4, 1995, pp. 1942-1948.
20. V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, NY, 2000.
21. L. Breiman, "Random forests," *Machine Learning*, Vol. 45, 2001, pp. 5-32.
22. C.-Z. Yang, C.-C. Hou, W.-C. Kao, and I.-X. Chen, "An empirical study on improving severity prediction of defect reports using feature selection," in *Proceedings of the 19th Asia-Pacific Software Engineering Conference*, 2012, pp. 240-249.
23. I. Herraiz, D. M. German, J. M. Gonzalez-Barahona, and G. Robles, "Towards a simplification of the bug report form in Eclipse," in *Proceedings of the 5th International Working Conference on Mining Software Repositories*, 2008, pp. 145-148.
24. C. J. V. Rijsbergen, S. E. Robertson, and M. F. Porter, *New Models in Probabilistic Information Retrieval*, British Library, London, 1980.
25. R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," *Journal of Machine Learning Research*, Vol. 9, 2008, pp. 1871-1874.
26. I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, Elsevier Inc., Morgan Kaufmann, Cambridge, MA, 2017.



Cheng-Zen Yang received the BS and MS degrees from Department of Computer Science and Information Engineering, National Chiao Tung University, Taiwan, in 1988 and 1990, respectively. Then he received his Ph.D. degree from Department of Computer Science and Information Engineering, National Taiwan University in 1996. Currently, he is an Associate Professor in Yuan Ze University. His research interests include software engineering, machine learning, text mining, and high-speed networking.

Kun-Yu Chen received the MS degree from Department of Computer Science and Engineering, Yuan Ze University, Taiwan, in 2014. His research interests include software repository mining, information retrieval, and text mining.

Anh-Hien Dao received the bachelor's degree from Hung Yen University of Technology and Education, and the master's degree from the VNU University of Engineering and Technology, Vietnam in 2007 and 2014, respectively. He received the Ph.D. degree from Department of Computer Science and Engineering, Yuan Ze University, Taiwan in 2021. Currently, he is a Lecturer at Hung Yen University of Technology and Education. His research interests include software engineering, software repository mining, and machine learning.