Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 5102897, 20 pages
https://doi.org/10.1155/2020/5102897

Research Article

High-Dimensional Hybrid Data Reduction for Effective Bug Triage

Xin Ge, Shengjie Zheng, Jiahui Wang, and Hui Li

The College of Information Science and Technology, Dalian Maritime University, Dalian 116026, China

Correspondence should be addressed to Xin Ge; ge_xin@dlmu.edu.cn

Received 28 November 2019; Revised 22 January 2020; Accepted 11 February 2020; Published 11 May 2020

Guest Editor: Chunlai Chai

Owing to the ever-expanding scale of software, solving the problem of bug triage efficiently and reasonably has become one of the most important issues in software project maintenance. However, there are two challenges in bug triage: low quality of bug reports and engagement of developers. Most of the existing bug triage solutions are based on the text information and have no consideration of developer engagement, which leads to the loss of bug triage accuracy. To overcome these two challenges, we propose a high-dimensional hybrid data reduction method that combines feature selection with instance selection to build a small-scale and high-quality dataset of bug reports by removing redundant or noninformative bug reports and words. In addition, we also study the recent engagement of developers, which can effectively distinguish similar bug reports and provide a more suitable list of the recommended developers. Finally, we experiment with four bug repositories: GCC, OpenOffice, Mozilla, and NetBeans. We experimentally verify that our method can effectively improve the efficiency of bug triage.

1. Introduction

A large amount of data is generated during software development and maintenance. Bug reports are generated continuously during this process. A bug report contains a basic description of the bug, error messages, current status of the bug (whether it has been solved or assigned to a developer), etc. [1]. According to statistics, bug fixing increases the costs of software companies by 45% [2] and increases the time of software development. Therefore, solving the problem of bug triage efficiently and accurately, that is, assigning the bug to the most suitable developer [3–9], becomes important for software projects. Bug tracking systems were developed to track all aspects of bug messages and help software developers fix bugs in time. These systems play an important role in dispatching work between developers. Typical bug tracking systems are Mantis [10], Bugzilla [11], and JIRA [12].

The earliest method of bug triage was to manually assign a bug to a corresponding developer. The senior manager would read a bug report in the bug tracking system and would select a suitable developer for this bug report based on his/her own experience and knowledge of the developer's ability. However, this method of assignment not only wastes time and human resources, but also has limited accuracy. First, in large-scale software development, the number of bugs dramatically increases, and the quality of bug reports may vary. In 2001–2010, 333,371 bugs were submitted to Eclipse from more than 34,917 developers. When developers submit many duplicated or invalid bugs, a lot of time is wasted [13–15]. Moreover, owing to a large number of developers, senior managers cannot remember the bug-fixing skills of each developer and the types of bugs that this developer is good at. Thus, senior managers may assign bugs improperly, which may reduce the accuracy of bug fixes. Recently, a novel approach based on software networks is proposed to address software quality-related problems, e.g., in the perspective of bug network [16] and social networks [17]. These methods need two essential premises, constructed network extracted from software and extra information such as the social relationship of developers. Due to these drawbacks, it is difficult to widely adopt to improve software quality and bug triage.

To solve the above problems of manual bug triage and improve classification accuracy, Murphy et al. [5] proposed to apply text categorization technology to bug triage; it automatically generated a list of recommended developers after training on a bug dataset with developer labels. In their

research, a bug report is an instance. The words in the bug report are the corresponding attributes. The developer is the label of an instance. Then, they apply the classification algorithm to predict the best developer sequence for bug fixes. Subsequently, some researchers used the vector space model to represent the bug report [2, 7, 18–24]. In addition, the theme model was used to identify different documents, and the words within the document have a specific relationship to each theme [25]. Researchers [26–31] not only used the theme model but also studied metadata (such as products, components, and operating system types in the bug report) to improve the accuracy.

Two challenges of automatic bug triage technology remain to be solved. First, the original bug repositories are large scale with low-quality bug reports. Many problems may arise due to large scale and low quality, such as extensive computations and a decline in predictive performance. Second, most of the existing work ignores the influence of developer engagement. Developers who have joined recently may be relatively more active because other developers may change positions or leave.

In this study, we propose a high-dimensional hybrid data reduction method to combine feature selection with instance selection method to build a small-scale and high-quality set of bug reports by removing redundant or noninformative bug reports and words. Our method combines feature selection (FS) with instance selection (IS) using the differential evolution (DE) method with updated rules of crossover and variation. In addition, we consider the developer engagement in each project for more reasonable bug triage. If the developer has dealt with bugs with this kind of product information before, we consider he/she is more active, and we are more likely to assign this bug report to him/her. We conduct experiments on four public bug repositories: GCC, OpenOffice, Mozilla, and NetBeans. The main contributions of this paper are summarized as follows:

(1) We present a high-dimensional hybrid data reduction method based on DE to obtain a small-scale and high-quality dataset for bug triage. Specifically, we aim to address two aspects: (a) to simultaneously reduce the bug reports dimension and the words dimension and (b) to improve the performance of bug triage.

(2) We consider the developer engagement level further and reorder the optimal list of recommended developers, combining it with product information that is related to bug reports and developers.

(3) We verify the effectiveness of our proposed method on four bug repositories (GCC, OpenOffice, Mozilla, and NetBeans). The experimental results show that the results obtained by using our proposed method are superior to the existing methods.

The rest of the paper is structured as follows. Section 2 presents closely related work on bug triage. Section 3 details the proposed DE method, which is improved in our paper. Section 4 describes our experiment with GCC, OpenOffice, Mozilla, and NetBeans and explains the results. Finally, Section 5 concludes our paper and describes future work.

2. Related Work

2.1. Methods of Bug Triage. The main purpose of bug triage is to find the right developer to fix a newly submitted bug [2, 7, 11, 19, 20, 32–34]. At present, machine learning has been used to accomplish automatic bug dispatching. When we use machine learning, the developer is considered as the category tag of the bug, and the text information of the bug is regarded as a feature. First, the algorithm learns on historical bug data. Then, the algorithm can predict which developer should be assigned to a new bug report. Murphy and Cubranic [7] propose a text classification approach to bug triage. Based on the Naïve Bayes classification algorithm, a list of recommended developers was predicted for Eclipse. Anvik et al. [2] used a variety of machine learning algorithms (such as Naïve Bayes, SVM, and C4.8) to learn historical data for bug triage. Tamrawi et al. [23] proposed "Bugzie," a method in which a fuzzy set was used to simulate the developer's expertise to determine whether a new bug report is suitable for this developer. Naguib et al. [35] used a latent Dirichlet allocation language model to learn the similarity between new bug reports and developers by learning from previously available data on bugs already fixed by developers. Yang et al. [27] divided bug reports into different topics. Zhang et al. [36] considered both developer relationships and topic models to recommend the bug report to a developer who would fix it better. Jeong et al. [21] used a Markov chain-based graph model to improve bug triage accuracy and tested it on Eclipse and Mozilla. Xia et al. [37] proposed a composite method named DevRec and analyzed bug reports and developers simultaneously.

Except for the bug report features, other information is also used to classify bugs. Linares-Vásquez et al. [38] analyzed the title comments of related documents and found relevant documents to recommend developers. Bhattacharya and Neamtiu [22] studied the tossing graph with various attributes, which included the developers who could not fix this bug and others who could fix it, to improve the triage accuracy. Kevic et al. [39] assigned the current bug to a developer by finding a developer whose fixed bugs were similar to the current bug report. Wang and Lo [40] used a new approach named FixerCache, which assigned new bug reports to developers based on the developer's enthusiasm for different product components. In the study by Shokripour et al. [41], four information resources were considered to obtain a list of recommended developers. Xia et al. [37] proposed the TopicMinerMTM model, which used training data to model the topic distribution of bug reports. However, Xia's approach had difficulties with distinguishing similar developers.

2.2. Bug Triage and Software Networks. Besides the bug reports, researchers adopt more information to address the bug-related problems. Because the quality of a software system is partially determined by its structure (topological structure), software systems can be modeled as complex networks in which software components are abstract nodes and their interactions are abstract edges. Therefore,

researchers have proposed many approaches and measures based on general software networks. Zhang and Lee [42] proposed an automated developer recommendation approach for bug triage via building the concept profile (CP) for extracting the bug concepts with topic terms from the documents produced by related bug reports. They identify and rank the important developers by using social network (SN) for bug fixing. Because of the functional form of the incoming link distribution of software dependence network, software is fragile with respect to the failure of a random single component. Locating a faulty component is easy if the failure only affects its nearest neighbors, while it is hard if it propagates further. Challet and Lombardoni [43] addressed the issue of how software components are affected by the failure, and the inverse problem of locating the faulty component through adopting bug propagation and debugging in asymmetric software structures. Pan et al. [44] also analyzed the bug propagation process based on the weighted software networks (WSNs). It considered the process of a bug in one class propagating to the other and is effect to locate the source of component with bug. The aforementioned approaches take advantage of complex networks, especially the relationship among developers, users, software components, and bug reports. In the perspective of software networks, bug triage involves developers and bug reports, and thus, the improvement of the accuracy of bug triage is able to promote and enhance the effectiveness of software network-related methods. For example, bug triage can help us gain a deeper insight of bug propagation on the software networks. Correspondingly, the novel measurements based on software networks may provide more useful information for bug triage. In other words, they are mutually beneficial.

In summary, we find that these traditional feature selection approaches do not have sufficient search range and depth and do not consider chronological order. Therefore, the denoising ability is limited, and the effect on data reduction is not obvious. This study addresses these two aspects. First, we fully consider the chronological order of bug reports. Second, we propose to improve the differential evolution method based on this to expand the search range and search depth. Meanwhile, we also can ensure the convergence of the method, and the accuracy of bug triage increases effectively. Unlike the above research, our paper focuses more on developer engagement. Except for the text provided in the bug report itself, the bug report and the developer's developers' bug-fixing experience are also used to obtain the best list of recommended developers. When the similarity of different bug reports is very high, our approach helps the classifier to strictly divide different products to complete the bug triage task successfully and substantially improve the efficiency of bug triage.

3. The Proposed Algorithm

3.1. Overview. In this section, we propose a high-dimensional hybrid data reduction method for bug triage, as shown in Figure 1. The method includes three main phases: data preprocessing, data reduction, and developer engagement detection. In the data preprocessing phase, we preprocess each bug report using word segmentation, stop words, stemming, and vector space representation to obtain the word vectors. In the data reduction phase, we propose a high-dimensional hybrid data reduction method to combine feature selection with the instance selection method to build a small-scale and high-quality dataset of bug reports by removing redundant or noninformative bug reports and words. In the developer engagement detection phase, we consider the influence caused by the product information of the current test case, which can include the level of developer engagement in the final order of recommended developers. These three models are described in the following sections.

3.2. Data Preprocessing. After extracting text and developer information from the bug report tracker, our method preprocessed the data to obtain word vectors for each bug report. The text preprocessing includes tokenization, stop-word removal, stemming, and keyword vector representation [11]. Specifically, tokenization converts text from the original bug report into a set of words. Stop-word removal refers to removing insignificant words that appear frequently in bug reports (such as "the" or "in"). Stemming transforms words that may appear in different forms into their basic form (for example, "computerized" can be changed into "computer"). Keyword vector representation produces a word vector to model a bug report after previous steps, and we delete words with the word frequency less than 10. Finally, our method uses a multidimensional vector space, where each word represents a dimension to describe the processed bug report collection. Every bug report is a vector based on the word dimensions, as shown in the following equation:

$$R = w_1, w_2, w_3, \ldots, w_i, \ldots, w_n, \quad i \in [1, n], \tag{1}$$

where R is a bug report and w_i refers to i word in the word dimension.

3.3. Data Reduction. After using the bug report preprocessing model, our method obtains word vectors for bug reports. Considering that some bug reports will become outdated over time, the data will be noisy and redundant. To reduce the impact of chronological order, our method divides the bug reports into three parts according to the ratio of 7 : 1 : 2 from small ID to large ID (the first 70% is the training set, the middle 10% is the verification set, and the last 20% is the test set). In the search process, our method first uses the training set for training and the verification set for examining to find the optimal solution results with FS, IS, and FS + IS. Next, the combination of the training set and validation set is regarded as a new training set, and our method obtains a final list of Top-10 developers for the test set. Subsequently, we will explain the rules of the DE method and the data reduction method of FS, IS, and FS and IS simultaneously (FS + IS).

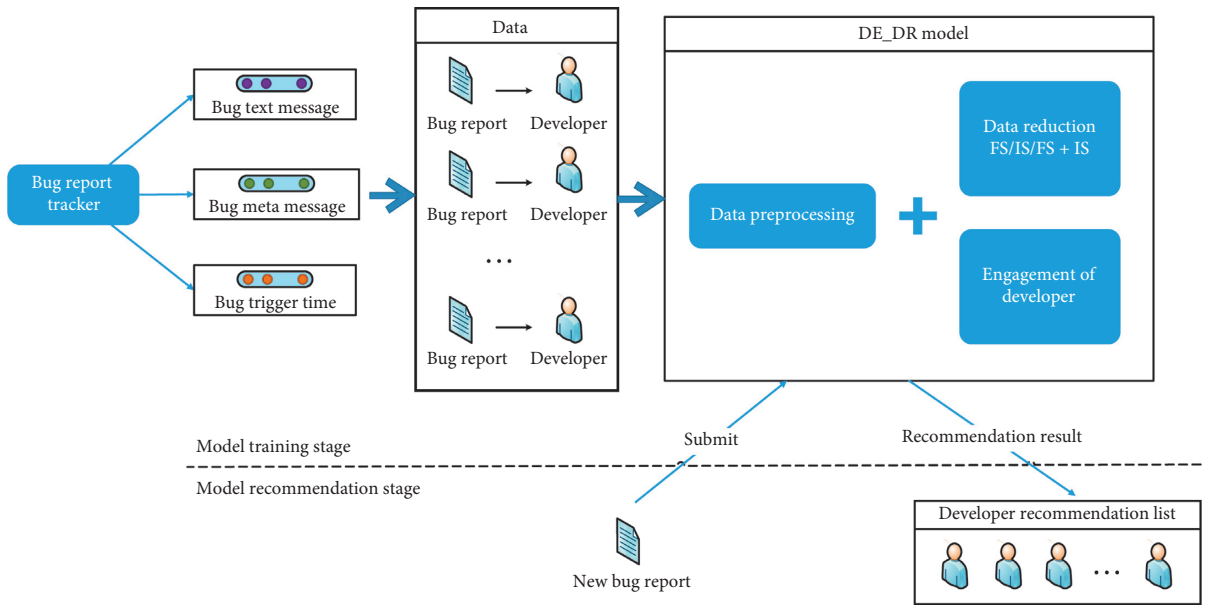

Figure 1: Flow diagram of bug triage process.

3.3.1. Population Coding. To represent feature combination and instance combination clearly and intuitively, we adopt binary coding: our method uses a feature vector with n feature dimensions to replace the feature sequence in the population, and it transforms the selected feature combinations into one 0, 1 binary string by the following equation:

$$F_i = f_1, f_2, f_3, \ldots, f_i, \ldots, f_N, \quad i \in [1, N], \tag{2}$$

where F_i is a feature vector corresponding to a feature sequence, $f_i = 0$ indicates that feature i is not selected and $f_i = 1$ indicates that feature i is selected, and N is the total number of features. Thus, a binary string represents a feature selection method.

Similarly, the sequence of instances in the dataset can be represented as a feature vector of $1 \times M$, and the selected instance combination is also a 0, 1 binary string, which is described by the following equation:

$$S_j = s_1, s_2, s_3, \ldots, s_j, \ldots, s_M, \quad j \in [1, M], \tag{3}$$

where S_j is a feature vector corresponding to an instance sequence, $s_j = 0$ indicates that instance j is not selected and $s_j = 1$ indicates that instance j is selected, and M is the total number of instances. A binary string represents an instance selection method.

3.3.2. Population Initialization. The population initialization in DE_IS (initial feature selection scheme): we sort bug reports by their IDs, extract all features to obtain a feature set, and generate 10 initial selection options that select 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 100%, respectively.

The population initialization in the initial feature selection scheme (DE_IS): we generate a random number between [0, 1] for each position in the binary string of each scheme in the instance selection scheme. If the number is less than 0.5, this position is set to 0. Otherwise, it is set to 1. In this case, our method can obtain ten initial instance selection schemes.

The initial extraction scheme in DE_(FS + IS) (combined scheme of feature selection and instance selection): first, our method generates 10 initial schemes according to the initial population generation methods in DE_FS and DE_IS; second, the generated features and instances are combined with the corresponding number of binary strings. Then, our method obtains 10 extraction schemes in the initial population.

For each individual in the initial population, after calculating their values of fitness, our method records the maximum value and its corresponding code value.

3.3.3. Genetic Manipulation

(1) Variation (Differential Variation). First, our method generates a variation rate P_{rv} randomly between [0, 1] and defines the differential variation rate as P_d (P_d in this method is always dynamic; it is judged according to the number of iterations that our method is inclined to random variation or differential variation, which takes place between the individual and the currently optimal individual. This is described by the following equation:

$$P_d = \text{de}\left(\frac{1 - \text{index}}{\text{total Generation}}\right), \tag{4}$$

where de is the differential coefficient of variation, index is the current iteration number, and total Generation is the number of iterations. If $P_{rv} < P_d$, we use this extraction scheme as a parent to carry out differential variation. Otherwise, we do not perform variation. In differential variation, Variation num (the number of mutated genes) positions of variant genes (L_v) are randomly selected in the

scheme, and differential variation rule is defined by the following equation:

$$L_n = \begin{cases} 1, & L_v = L_b \text{ or } (L_v \neq L_b \text{ and } D_r \geq 0.5), \\ 0, & L_v \neq L_b \text{ and } D_r < 0.5, \end{cases} \quad (5)$$

where L_v is the current variant gene position, L_b is the optimal individual corresponding gene position, L_n is the corresponding gene position of the new individual, and D_r is a random number. If L_v is different from L_b, L_n is 1; if not, a random number D_r between [0, 1] is generated. If $D_r \geq 0.5$, then L_n is 1; otherwise, it is set to 0. The other genetic positions of the new individual are identical to the currently selected parent.

(2) Crossover. Similarly, a crossover probability P_{rc} is randomly generated at start. The crossover variation rate is defined as P_c (also dynamically changed), and the formula is shown in the following equation:

$$P_c = \text{aberrance Rate}\left(\frac{\text{index}}{\text{total Generation}}\right), \quad (6)$$

where aberrance Rate is the cross coefficient of variation, index is the current iteration number, and totalGeneration is the number of iterations. If $P_{rc} < P_c$, cross-variation is performed to create a child generation. When cross-variation works, two positive integers P_1 and P_2 are randomly generated ($P_1 < P_2$ and $P_1, P_2 \in [1, \text{length}]$), then the code between P_1 and P_2 is cross-operated according to the midpoint MID, where MID = $(P_1 + P_2)/2$. The rest of the new individual is the same as the parent.

(3) Variation (Random Variation). For each extraction scheme of population, our method generates a variation rate P_{rv} and defines the random variation rate as P_r (also dynamically changed); its formula is defined as follows:

$$P_r = \text{heredity Date}\left(\frac{1 - \text{index}}{\text{total Generation}}\right), \quad (7)$$

where herediyDate is the random variation coefficient, index is the current iteration number, and totalGeneration is the number of iterations. When P_{rv} is less than P_r, the extraction scheme mutates; otherwise, it is not mutated. During the random variation, Variation num positions of variant genes (L_v) are randomly selected in the binary string, and the random variation rules are described as follows:

$$L_n = \begin{cases} 1, & L_v = 0, \\ 0, & L_v = 1, \end{cases} \quad (8)$$

where L_v is the current variant gene position and L_n is the corresponding gene position of the new individual. If its value is 0, it will be changed to 1, and the other condition is the opposite.

(4) Selection. We calculate the fitness values of all individuals in the population. Our method not only conserves the new individuals that are generated by the crossover, but also the parent individuals. Subsequently, a new population with twice the size is aggregated. Next, all individuals are sorted according to their fitness value in descending order. The former half is selected, and the latter half is eliminated. Meanwhile, the binary code that has optimal fitness is updated.

Table 1: Parameters.

Parameter	Meaning
Train	The training set
NP	Population number
N	The number of features
M	The number of bug reports
Variation num	The number of mutated genes
totalGeneration	The number of iterations
P_d	The differential variation rate
P_c	The crossover variation rate
P_r	The random variation rate

3.3.4. Data Reduction Algorithm. After our analysis, both FS and IS can achieve the goal of extracting useful attributes. FS can be reduced at the attribute level, and IS can be reduced at the instance level. The effect of FS + IS might surpass the result obtained by using FS and IS separately. Thus, we choose three feature extraction methods: FS, IS, and FS + IS.

The parameters used in this section are defined in Table 1.

$$P_d = \text{de}\frac{1 - \text{index}}{\text{totalGeneration}}, \quad \text{de} = \alpha, \quad (9)$$

$$P_c = \text{aberranceRate}\left(\frac{\text{index}}{\text{totalGeneration}}\right), \quad \text{aberranceRate} = \beta, \quad (10)$$

$$P_r = \text{hereditDate}\frac{1 - \text{index}}{\text{totalGeneration}}, \quad \text{hereditDate} = \gamma. \quad (11)$$

Algorithm 1 indicates that we carry out feature selection (DE_FS) or instance selection (DE_IS) on the dataset. In the first line, we empty the initial extraction scheme and the best best_individual scheme. In lines 2–6, we judge whether it is a feature selection or an instance selection, and we call Algorithm 2 for the former and Algorithm 3 for the latter. In lines 7-8, we calculate the fitness value of each extraction scheme in population through the fitness function (the accuracy of Naïve Bayesian Mode (NBM)) and record the extraction scheme with the largest fitness value. For each of these three variants, parent can only choose one method to mutate and generate a child. Lines 13–26 are the choice of differential variation. And lines 28–36 present the choice of cross-variation. The choice of random variation is in lines 38–44. In lines 49–54, we unite the parents and child generated, sorting the fitness in descending order, select the best Top 10 to enter the next generation, and update the optimal fitness and the corresponding binary code. In line 57, after executing T iterations, we decode the saved optimal extraction scheme into a corresponding dataset

Input: $Train$, NP, T, N, M, $heredityDate$, $aberranceRate$,de
Output: R$educed_Train$

(1) $Population \longleftarrow \varnothing best\_individual \longleftarrow \varnothing$
(2) **if** $feature\ select$ **then**
(3) $Population \longleftarrow Initialize\_FS(Train, NP, N)$
(4) **else**
(5) $Population \longleftarrow Initialize\_IS(Train, NP, M)$
(6) **end if**
(7) $Calculate\ the\ fitness\ value\ of\ each\ individual\ by\ NBM$
(8) $best\_individual \longleftarrow The\ individual\ with\ the$ largest $fitness\ value$
(9) $T = 1$
(10) **while** $T < totalGeneration$ **do**
(11) **for** $all\ I\ from$ 1 $to\ N$ **do**
(12) $//Difference$ variation
(13) **if** $randomf(0, 1) < de(1 - index/totalGeneration)$ **then**
(14) $Randomly\ generate\ Variation\_Num\ mutation\ location$
(15) $Generates\ a\ new\ individual\ exactly\ the\ same\ as\ the\ current\ selected\ parent$
(16) **for** $all\ j\ from$ 1$toVariation_Num$ **do**
(17) **if** $father[j] \neq best[i]$ **then**
(18) $kid[i] = 1$
(19) **else**
(20) **if** $randomf(0, 1) < 0.5$ **then**
(21) $kid[i] = 0$
(22) **else**
(23) $kid[j] = father[j]$
(24) **end if**
(25) **end if**
(26) **end for**
(27) $//Crossover$
(28) **else**
(29) **if** $randomf(0, 1) < aberranceRate(index/totalGeneration)$ **then**
(30) $//Generatesanewindividualexactlythesameastheselectedparent$
(31) **if** $featureselect$ **then**
(32) $P1andP2arerandomlygeneratedfrom0toN(P1 < P2)$
(33) **else**
(34) $P1andP2arerandomlygeneratedfrom0toM(P1 < P2)$
(35) $P1andP2areswappedinsectionsaccordingtothemidpo$int
(36) **end if**
(37) $//Random$variation
(38) **else**
(39) **if** $randomf(0, 1) < heredityDate(1 - index/tnoqtha_lxG7eCn; eration)$ **then**
(40) $Generatesanewindividualexactlythesameastheselectedparent$
(41) $RandomlygenerateVariation\_Nummutationlociation$
(42) **for** $all\ j\ from$1$toVariation_Num$ **do**
(43) $Ifthelocusis0, setitto1, andifitis1thensetitto0$
(44) **end for**
(45) **end if**
(46) **end if**
(47) **end if**
(48) **end for**
(49) $Calculate\ fitness\ values\ for\ all\ individuals$
(50) $Select\ theTop10\ from\ all\ individuals\ sorted\ by\ fitness\ value\ from$ large $to\ small$
(51) **if** $the\ current\ optimal\ value > the\ original\ optimal\ value$ **then**
(52) $update\ the\ current\ optimal\ value\ and\ the\ corresponding\ binary\ code$
(53) **end if**
(54) $Updating\ Population$
(55) $T++$
(56) **end while**
(57) $The\ binary\ string\ corresponding\ to\ best\_individual\ is\ decoded\ as$ R$educed_Train$
(58) **return** R$educed_Train$

Algorithm 1:Reduction (DE).

```
    Input: Train, NP, N
    Output: Initialize_Population_FS
(1)  Initialize_Population_FS←∅
(2)  metrics←[IG]
(3)  for each metric in metrics do
(4)    Order featuresbymetric
(5)    for alli from1 to10 do
(6)      Take the first i × 10 of N as fs
(7)      Initialize_Population_FS.add(fs)
(8)    end for
(9)  end for
(10) return Initialize_Population_FS
```

Algorithm 2: Initialize_FS.

```
    Input: Train, NP, M
    Output: Initialize_Population_IS
(1)  Initialize_Population_IS←∅
(2)  for alli from 1 to NP do
(3)    Treat individual i as ins
(4)    for ins gene location j from 1 to M do
(5)      rate = Random(0, 1)
(6)      if rate ≥ 0.5 then
(7)        Change the j locus of the ith individual to 1
(8)      else
(9)        Change the j locus of the ith individual to 0
(10)     end if
(11)   end for
(12) end for
(13) Initialize_Population_IS.add(ins)
(14) return Initialize_Population_IS
```

Algorithm 3: Initialize_IS.

Reduced_Train. In line 58, we return Reduced_Train, which is reduced (if this gene position is different from the corresponding position of an optimal individual, the same position of the new individual is 1. Otherwise, a random number between [0, 1] is generated. If this number is equal or greater than 0.5, the position of the new individual is 1; otherwise, it is set to 0. The other genetic positions of the new individual are identical to the currently selected parent).

Algorithm 2 represents the population initialization process when we select features using the DE method. In the first line, we empty the initial extraction scheme. In line 2, the IG is used to select features and is recorded as metrics. In lines 3–9, we sort the training set from high to low according to the importance of features and select top 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 100% to add them into the feature selection scheme (Initialize_Population_FS). We set the extracted feature column to 1 and the opposite feature column to 0.

Algorithm 3 represents the population initialization process when we select instances using the DE method. In the first line, we empty the initial extraction scheme. In lines 2–11, each instance selection scheme is represented as a $1 \times M$ binary string. Then, we generate a random decimal between 0 and 1 for each gene position of each extraction scheme. If the number is greater than or equal to 0.5, the corresponding gene position of the individual is 1. Otherwise, it is set to 0. We add generated instance selection schemes to Initialize_Population_IS. In line 14, we return all generated extraction schemes.

DE_(FS + IS) indicates that we select features and instances simultaneously for the dataset using the DE method. In line 1, we empty the initial extraction scheme and the best best_individual scheme. In lines 2–6, we combine the initialized feature selection scheme with the initialized instance selection scheme. In lines 7-8, we calculate the fitness value of each extraction scheme to save the best extraction scheme and its fitness value. Differential variation is described in lines 11–26. Cross-variation is detailed in lines 28–37. Lines 39–47 present the random variation. In lines 51–56, parents and children are mixed and sorted by the fitness values to select the best Top 10, which can enter the next generation. The optimal fitness and binary code are updated. In line 60, after T iterations, we obtain Reduced_Train by decoding the best extraction scheme. In line 61, we return the reduced Reduced_Train.

3.4. Engagement Detection. After analyzing the actual situation, we find that some developers may change jobs or leave the company over time. Meanwhile, a bug report that has the same problem with different product information may be very similar. Thus, we reorder developers depending on their level of engagement and the product information. First, we find the corresponding product information in the latter N bug reports of each developer's training set (the original training set and the validation set, which analyze developer engagement using recent bug reports). Subsequently, we use a linked list to encapsulate the data to store more product information. Next, the NBM classifier is used to select the Top 30 developers. We record the product information of current bug reports and check whether each developer has dealt with the same product information or not. If not, we choose to discard this developer, and the subsequent developers will fill up the empty positions one by one. Finally, we can select the optimal Top 10 in this way (Algorithm 4).

4. Experiment

We conduct several experiments to verify our method. In Section 4.1, we describe the experimental design, including data preparation and experimental setup. In Section 4.2, we specify the evaluation metrics. In Section 4.3, we discuss experimental results, which answer our research questions.

4.1. Experimental Design

4.1.1. Data Preparation. To demonstrate the effectiveness of our approach, we carry out a series of experiments on four large-scale open-source bug repositories: GCC, OpenOffice, NetBeans, and Mozilla. We only collected the fixed bug reports which were denoted as "resolved" or "closed" before December 31, 2014, due to their stability and reliability. We have uploaded our experimental datasets to URL (https://github.com/gexinxinge/complexnetwork). Table 2 shows the dataset statistics, including the total number of products, total number of reports, total number of words, total number of developers, and time range.

We use the processing method from the previous research [23, 26]. The label of the developer who was assigned to the bug report is regarded as the developer who fixed this bug. To obtain a list of recommended developers, we exclude the bugs that were never fixed [2]. Moreover, data on developers who worked on a small number of bug reports would not be helpful for obtaining accurate results; therefore, we exclude the developers who worked on less than 10 bug reports from our study.

4.1.2. Experimental Setup. In this paper, we segment data by a chronological method. The bug reports are divided into three parts in chronological order. Seventy percent is used as the training set, ten percent is used as the validation set, and the remaining twenty percent is used as the test set. We make the following adjustments to the parameters (the experimental parameters of the three differential methods are consistent): $\alpha = 0.7, \beta = 0.08, \gamma = 0.8$. It is worth pointing out that our approach is sensitive to these parameters. In equation (9), α makes an influence on differential variation rate (Pd). In equation (10), β decides the value of crossover variation rate (Pc). And in equation (11), γ affects the random variation rate (Pr). For example, if the values of α and γ are too large, it is hard to remove unimportant features. However, small values may lead to the local optimum. Therefore, we use reasonable values published by others' experience using difference algorithm. In this way, our method can not only have sufficient search scope, but also converge in the end.

We use two types of benchmarks for comparison: supervised methods and unsupervised methods. Supervised benchmark methods mostly include classifiers that are effective in text classification: Naïve Bayes (NB), polynomial Naïve Bayes (NBM), support vector machine (SVM), k-nearest neighbor (KNN), random tree (RT), and decision tree (J48).

4.2. Evaluation Metrics. In this paper, we use the accuracy of Top-k developers to evaluate the quality of bug triage, that is, the ratio of the predicted exact quantity to all test data. The accuracy of Top-1 to Top-10 is examined in the experiment altogether. The accuracy of Top-k is calculated by the following equation:

$$\text{Accuracy}_k = \frac{\text{Bug\_}k_{\text{correct}}}{\text{Bug}_{\text{total}}}, \tag{12}$$

where Accuracy_k is the accuracy of Top-k, $\text{Bug\_}k_{\text{correct}}$ refers to the number of bug reports that are assigned correctly to Top-k, and $\text{Bug}_{\text{total}}$ is the total number of bug reports in Top-k. A higher Accuracy_k describes a better list of recommended developers.

4.3. Experimental Results

4.3.1. RQ1: Which Supervised Learning Algorithms Are the Most Suitable? In this experiment, six supervised learning algorithms (NB, NBM, SVM, KNN, RT, and J48) are applied to classify four open-source bug repositories (Mozilla, NetBeans, OpenOffice, and GCC). We use the classification accuracy as an evaluation metric to choose the best classification algorithm.

The results are shown in Tables 3–6. According to the results, the classification effect of NBM (which is presented in bold) is the best among the six methods. The highest classification accuracies of NBM on four bug repositories are 28.89%, 57.59%, 48.39%, and 68.84%, respectively. Moreover, NB ranks second; its highest classification accuracies are 23.69%, 54%, 37.94%, and 50.27%, respectively. Both methods have much higher classification accuracy than other classifiers.

For OpenOffice, the accuracies of NBM from Top-1 to Top-10 are much higher than those of other algorithms (NB, NBM, SVM, KNN, RT, J48). Additionally, the Top-10 accuracy of NBM is 37.33% higher than the Top-10 accuracy of SVM on OpenOffice. Similarly, Table 6 shows that NBM performs the best among six approaches (NB, NBM, SVM,

Input: $Train$, NP, T, N, M, $heredityDate$, $aberranceRate$, de
Output: $Reduced\_Train(FS + IS)$

(1) $Population \leftarrow \varnothing$ $best\_individual \leftarrow \varnothing$
(2) **for** $all\ i from 1 to\ NP$ **do**
(3) $Initialize\_Population\_FS = Initialize\_FS(Train, NP, N)$
(4) $Initialize\_Population\_IS = Initialize\_IS(Train, NP, M)$
(5) $Population \leftarrow Combine(Initialize\_Population\_FS, Initialize\_Population\_IS)$
(6) **end for**
(7) $Calculate\ the\ fitness\ value\ of\ each\ individual\ by\ NBM$
(8) $best\_individual \leftarrow The\ individual\ with\ the\ largest\ fitness\ value$
(9) **while** $T < totalGeneration$ **do**
(10) **for** $all\ i from\ 1\ to\ NP$ **do**
(11) $//Difference\ variation$
(12) **if** $randomf(0, 1) < de(1 - index/tnoqtha_lxG7eCn; eration)$ **then**
(13) $Randomly\ generate\ Variation\_Num\ mutation\ location$
(14) $Generates\ a\ new\ FS\ individual\ exactly\ the\ same\ as\ the\ selected parent$
(15) $Generates\ a\ new\ IS\ individual\ exactly\ the\ same\ as\ the\ selected parent$
(16) **for** $all j from\ 1\ to\ Variation\_Num$ **do**
(17) **if** $father[j] \neq best[i]$ **then**
(18) $kid[i] = 1$
(19) **else**
(20) **if** $randomf(0, 1) < 0.5$ **then**
(21) $kid[i] = 0$
(22) **else**
(23) $kid[j] = father[j]$
(24) **end if**
(25) **end if**
(26) **end for**
(27) $//Crossover$
(28) **else**
(29) **if** $randomf(0, 1) < aberranceRate(index/tnoqtha_lxG7eCn; eration)$ **then**
(30) $Generates\ a\ new\ FS\ individual\ exactly\ the\ same\ as\ the\ selected parent$
(31) $Generates\ a\ new\ IS\ individual\ exactly\ the\ same\ as\ the\ selected\ parent$
(32) $//Cross\_feature\ select$
(33) $P1\ and\ P2\ are\ randomly\ generated\ from\ 0\ to\ N(P1 < P2)$
(34) $P1\ and\ P2\ are\ swapped\ in\ sections\ according\ to\ the\ midpoint$
(35) $//Cross\_instance\ select$
(36) $P1\ and\ P2\ are\ randomly\ generated\ from\ 0\ to\ M(P1 < P2)$
(37) $P1\ and\ P2\ are\ swapped\ in\ sections\ according\ to\ the\ midpoint$
(38) $//Random\ variation$
(39) **else**
(40) **if** $randomf(0, 1) < heredityDate(1 - index/tnoqtha_lxG7eCn; eration)$ **then**
(41) $Generates\ a\ new\ IS\ individual\ exactly\ the\ same\ as\ the\ selected parent$
(42) $Generates\ a\ new\ IS\ individual\ exactly\ the\ same\ as\ the\ selected parent$
(43) $Randomly\ generate\ Variation\_Num\ mutation\ lociation$
(44) **for** $all j from 1\ to Variation\_Num$ **do**
(45) $If\ the\ locus\ is\ 0, set\ it\ to\ 1, and\ if\ it\ is\ 1\ then\ set\ it\ to\ 0$
(46) **end for**
(47) **end if**
(48) **end if**
(49) **end if**
(50) **end for**
(51) $Calculate\ fitness\ values\ for\ all\ individuals$
(52) $Select\ the\ Top10\ individuals\ sorted\ by\ fitness\ value$
(53) $Enter\ the\ next\ generation\ and\ eliminate\ the\ rest.$
(54) **if** $the\ current\ optimal\ value > the\ original\ optimal\ value$ **then**
(55) $update\ the\ current\ optimal\ value\ and\ the\ corresponding\ binary\ code$
(56) **end if**
(57) $Updating\ Population$
(58) $T++$
(59) **end while**
(60) $The\ binary\ string\ corresponding\ to\ best\_individual\ is\ decoded\ as\ Reduced\_Train(FS + IS)$
(61) **return** $Reduced\_Train(FS + IS)$

Algorithm 4: DE_(FS + IS).

Table 2: Statistics of datasets.

	Product	Bug reports	Words	Developers	Period
GCC	2	13961	51005	263	1999/08/03–2014/12/01
OpenOffice	37	23475	50166	553	2000/10/21–2014/12/31
Mozilla	12	18793	33186	1022	1999/03/17–2014/12/31
NetBeans	12	21538	41256	265	1999/02/11–2014/12/31

Table 3: The accuracy of Mozilla using different classifiers.

Mozilla	NB	NBM	SVM	KNN	RT	J48
Top-1	0.0815	**0.0906**	0.0226	0.0509	0.046	0.0997
Top-2	0.1143	**0.1463**	0.0226	0.0509	0.046	0.1334
Top-3	0.1369	**0.1833**	0.0352	0.0627	0.0582	0.1477
Top-4	0.1578	**0.2045**	0.0352	0.0634	0.0582	0.1547
Top-5	0.1749	**0.2233**	0.0387	0.0662	0.0613	0.1599
Top-6	0.1868	**0.2404**	0.0397	0.0672	0.0624	0.1624
Top-7	0.1976	**0.2551**	0.0397	0.0672	0.0624	0.1652
Top-8	0.2129	**0.2669**	0.0397	0.0672	0.0624	0.1693
Top-9	0.2265	**0.278**	0.0397	0.0672	0.0624	0.1704
Top-10	0.2369	**0.2889**	0.0397	0.0672	0.0624	0.1704

Table 4: The accuracy of NetBeans using different classifiers.

NetBeans	NB	NBM	SVM	KNN	RT	J48
Top-1	0.2157	**0.2917**	0.1478	0.1184	0.1184	0.2163
Top-2	0.3041	**0.3972**	0.1478	0.1184	0.1184	0.2783
Top-3	0.3649	**0.4311**	0.1478	0.1184	0.1184	0.2979
Top-4	0.4039	**0.4612**	0.1548	0.1251	0.1251	0.3069
Top-5	0.4359	**0.4847**	0.1548	0.1254	0.1257	0.317
Top-6	0.4572	**0.5091**	0.1585	0.1302	0.1307	0.3212
Top-7	0.4825	**0.5273**	0.2508	0.2202	0.2205	0.3621
Top-8	0.5018	**0.5447**	0.2508	0.2202	0.2205	0.3933
Top-9	0.5195	**0.5627**	0.2508	0.2202	0.2205	0.4084
Top-10	0.54	**0.5759**	0.2508	0.2202	0.2205	0.4087

Table 5: The accuracy of OpenOffice using different classifiers.

OpenOffice	NB	NBM	SVM	KNN	RT	J48
Top-1	0.0969	**0.182**	0.0044	0.076	0.0716	0.132
Top-2	0.1607	**0.275**	0.0059	0.0775	0.0731	0.1984
Top-3	0.219	**0.3211**	0.0228	0.0935	0.091	0.2247
Top-4	0.249	**0.3507**	0.0434	0.1118	0.1087	0.2401
Top-5	0.2801	**0.3775**	0.0434	0.1128	0.1089	0.2583
Top-6	0.3054	**0.4008**	0.0491	0.117	0.1143	0.2666
Top-7	0.3274	**0.4209**	0.0505	0.1185	0.1155	0.2715
Top-8	0.3434	**0.4437**	0.0515	0.1221	0.1192	0.2745
Top-9	0.3603	**0.4636**	0.1018	0.1575	0.1577	0.2897
Top-10	0.3794	**0.4839**	0.1106	0.1594	0.1602	0.3066

Table 6: The accuracy of GCC using different classifiers.

GCC	NB	NBM	SVM	KNN	RT	J48
Top-1	0.1598	**0.3464**	0.3295	0.1286	0.1188	0.1821
Top-2	0.2464	**0.525**	0.3313	0.1304	0.1205	0.2759
Top-3	0.3125	**0.5786**	0.3313	0.1304	0.1205	0.2955
Top-4	0.3589	**0.6036**	0.3313	0.1304	0.1205	0.3009
Top-5	0.4	**0.6241**	0.3313	0.1304	0.1205	0.3027
Top-6	0.4259	**0.6464**	0.3321	0.1313	0.1214	0.3045
Top-7	0.4473	**0.6607**	0.3321	0.1313	0.1214	0.3045
Top-8	0.4679	**0.6705**	0.3402	0.1393	0.1295	0.3089
Top-9	0.4884	**0.6804**	0.3402	0.1393	0.1295	0.3107
Top-10	0.5027	**0.6884**	0.342	0.1411	0.1304	0.3116

KNN, RT, and J48). The Top-1 accuracy of NBM is 22.76% higher than that of RT and the Top-10 accuracy is 55.8% higher than RT on GCC.

Therefore, we conclude that NBM is the most effective algorithm among all methods. Thus, we use NBM as a reliable classification algorithm in subsequent experiments.

4.3.2. RQ2: Can Our Proposed FS and IS Methods Perform Better Than Traditional FS and IS Methods in terms of Data Reduction? In this part, we conduct four experiments on four open-source bug repositories, respectively. For each bug repository, we consider the data reduction of the original bug repository to specific data size, using feature selection or instance selection methods as benchmarks. We choose three traditional feature selection methods (IG, CHI, and SU) and three traditional instance selection methods (CNN, ENN, and ICF) as benchmarks to calculate Top-k accuracies, which reduce the original bug repository to the same data size as benchmarks. The accuracies of our proposed FS and IS heuristic search method with the same size of data reduction are also obtained. The column with the best effect is shown in bold in Tables 7–10.

Table 7 shows that our proposed FS method has 26.01% accuracy of Top-10 and the IS method has 30.44% accuracy of Top-10. Compared with the original data reduction and the traditional FS and IS methods, our proposed methods perform better for Mozilla. Meanwhile, the accuracy of IS is 4.43% higher than that of FS. The result for OpenOffice (Table 9) is quite similar to Mozilla. Our proposed FS method has the highest accuracy of 46.26% among feature selection methods, and the accuracy of our proposed IS method is 53.76%, which is also the best in IS approaches. The accuracy of IS is 7.5% higher than FS. Table 10 shows similar experimental results for GCC: the highest accuracy is 84.51% for our proposed FS method (in FS methods) and the highest accuracy is 83.69% for our proposed IS approach (in IS approaches). The only difference from the former is that FS is 0.82% higher than IS, which indicates that FS is more suitable for data reduction on GCC. However, there are many differences in NetBeans results (Table 8): our FS method is still the best with 54.14% accuracy among FS methods. However, a traditional IS approach called ENN has the highest accuracy of 59.88% in IS methods. Our IS method appears to overfit. The accuracy is rather high on the validation set (61%) but is reduced on the test set. A possible reason is that the developer flow of NetBeans may have been frequent recently. Moreover, NetBeans is not greatly affected by text information. In the RQ5, we verify this reason further.

We consider that when reducing to the same data size, the heuristic search method is generally better than the traditional IS and FS search method. The main reasons are as follows:

(1) The traditional FS methods and traditional IS methods are based on the overall training set. However, the effect of the bug report is time-sensitive. Outdated bug reports will add redundancy and noise, which may affect the classification accuracy. Traditional IS and FS methods cannot effectively eliminate noisy and redundant data.

(2) In the 7 : 1 : 2 search strategy, our proposed approach focuses more on extracting features related to recent bug reports, which is helpful for removing noisy and redundant data.

4.3.3. RQ3: Are the Accuracies of Our Proposed FS + IS, Best FS, and Best IS Methods Improved Comparing with the FS + IS Results of Xuan? If Our Results Are Improved, Which One Is the Most Effective among Our Three Methods? Based on the experimental results in RQ2, we reproduce the FS -> IS (IG + ICF) and IS -> FS (ICF + IG) experiments of Xuan and gain the Top-k accuracy on four bug repositories (Mozilla, NetBeans, OpenOffice, and GCC). In addition, we also combine our proposed FS method with the IS method (FS + IS) to reduce data of four bug repositories. The comparative results are shown in Tables 11–14. Compared with FS -> IS and IS -> FS, our proposed heuristic search method is better when reducing to the same data size.

Our FS + IS method has the best effect for Mozilla in Table 11 and NetBeans in Table 12. The accuracies are 29.44% and 56.07%, respectively. The Top-10 accuracy of FS + IS is 6.78% higher than that of IS -> FS on Mozilla and 16.42% higher than that of IS -> FS on NetBeans. We think another reason is that the heuristic search FS + IS focuses on the extraction of features related to recent bug reports. Thus, the amount of information loss is relatively small, and the ability to denoise and exclude redundancy is stronger. For OpenOffice in Table 12, the best accuracy 53.76% belongs to our IS method, which is 7.73% higher than IS -> FS and 45.57% higher than FS -> IS. On the contrary, we can find that our FS approach, which has 84.51% accuracy, is the most effective on GCC in Table 14. The Top-10 accuracy of FS is 9.23% higher than that of IS -> FS method on GCC.

We conclude that our proposed approaches are overall more effective than the results of Xuan. For four different bug repositories, FS + IS performs well on Mozilla and NetBeans. However, OpenOffice has the optimal effect with IS and the best of GCC is FS.

4.3.4. RQ4: How about the Data Reduction Effect of Our Proposed FS, IS, and FS + IS Methods? In this experiment, we use our proposed FS, IS, and FS + IS methods based on particle swarm optimization to obtain data reduction rates and study the effect of our approaches on four open bug repositories (Mozilla, NetBeans, OpenOffice, and GCC). The results are detailed in Tables 15–18 (data reduction degree of FS or IS and overall reduction accuracy refer to the ratio of the reserved instances or features to the original quantity). Here, the "original information" represents the dataset which is divided into 70% training dataset, 10% validation dataset, and 20% test dataset. Our approach finds primary features and instances using differential evolution algorithm with training set. Then, they are merged to get a new training set "original information (7 + 1)."

Table 7: Comparison of different feature selection and instance selection methods on Mozilla.

Mozilla	Original	IG	CHI	SU	FS
Top-1	0.0906	0.0723	0.0723	0.0723	0.0868
Top-2	0.1463	0.1155	0.1155	0.1155	0.13
Top-3	0.1833	0.1792	0.1792	0.1792	0.1578
Top-4	0.2045	0.1944	0.1944	0.1944	0.178
Top-5	0.2233	0.2072	0.2072	0.2072	0.1956
Top-6	0.2404	0.2196	0.2196	0.2196	0.2104
Top-7	0.2551	0.23	0.23	0.23	0.2287
Top-8	0.2669	0.2397	0.2397	0.2397	0.2424
Top-9	0.278	0.2487	0.2487	0.2487	0.2518
Top-10	0.2889	0.2563	0.2563	0.2563	0.2601
Mozilla	*Original*	*CNN*	*ENN*	*ICF*	*IS*
Top-1	0.0906	0.0152	0.0218	0.0225	**0.1208**
Top-2	0.1463	0.0218	0.0402	0.0506	**0.1884**
Top-3	0.1833	0.0277	0.0568	0.0654	**0.2105**
Top-4	0.2045	0.0377	0.0724	0.0841	**0.2351**
Top-5	0.2233	0.044	0.0845	0.1049	**0.2528**
Top-6	0.2404	0.0485	0.097	0.1195	**0.2639**
Top-7	0.2551	0.054	0.1118	0.1281	**0.276**
Top-8	0.2669	0.0578	0.124	0.1423	**0.2843**
Top-9	0.278	0.063	0.133	0.1569	**0.2933**
Top-10	0.2889	0.0661	0.1461	0.1676	**0.3044**

Table 8: Comparison of different feature selection and instance selection methods on NetBeans.

NetBeans	Original	IG	CHI	SU	FS
Top-1	0.2917	0.1778	0.1778	0.1778	0.2532
Top-2	0.3972	0.2571	0.2571	0.2571	0.3524
Top-3	0.4311	0.3188	0.3188	0.3188	0.4012
Top-4	0.4612	0.3551	0.3551	0.3551	0.4329
Top-5	0.4847	0.3877	0.3877	0.3877	0.4576
Top-6	0.5091	0.417	0.417	0.417	0.4732
Top-7	0.5273	0.4383	0.4383	0.4383	0.4911
Top-8	0.5447	0.4574	0.4574	0.4574	0.5072
Top-9	0.5627	0.4732	0.4732	0.4732	0.5251
Top-10	0.5759	0.4881	0.4881	0.4881	0.5414
NetBeans	*Original*	*CNN*	*ENN*	*ICF*	*IS*
Top-1	0.2917	0.0378	**0.225**	0.1863	0.2993
Top-2	0.3972	0.0614	**0.3384**	0.2706	0.3965
Top-3	0.4311	0.0867	**0.4038**	0.3353	0.4418
Top-4	0.4612	0.1084	**0.4512**	0.3828	0.4661
Top-5	0.4847	0.1268	**0.4928**	0.4203	0.4812
Top-6	0.5091	0.1405	**0.5216**	0.4468	0.4923
Top-7	0.5273	0.1554	**0.5462**	0.4689	0.5058
Top-8	0.5447	0.1707	**0.5662**	0.4883	0.5207
Top-9	0.5627	0.183	**0.5821**	0.5053	0.5299
Top-10	0.5759	0.1974	**0.5988**	0.5171	0.5409

According to our results, the reduction degrees of FS + IS are 96.05% on Mozilla, 94.43% on GCC, and 97.08% on OpenOffice, which can substantially reduce the data dimensions. However, FS + IS does not work best on any bug repository. For Mozilla, FS has the highest reduction degree of 97.21%, which is 1.16% higher than FS + IS. However, the reduction degrees of FS are smaller than for the FS + IS method on OpenOffice by 2.16% and GCC by 16.19%. The IS method also has a low degree of reduction on Mozilla, OpenOffice, and GCC. Nevertheless, it can reach 50% on OpenOffice. Moreover, our three methods do not perform well on NetBeans. If we compare the performance of three methods according to the degree of reduction, the result is FS + IS > FS > IS.

The degree of data reduction by FS, IS, and FS + IS based on the DE method is quite large. Compared with the traditional methods of removing invalid bug reports by name, the heuristic search based on the DE method, which can automatically delete invalid bug reports, is more effective in denoising and excluding redundancy. However, according to the previous experiment, FS + IS cannot achieve a further improvement in the accuracy of the NBM classifier compared to separate FS and IS methods. We consider the reason may be the excessive reduction, which can result in serious information loss. Therefore, we conclude that the data reduction of FS, IS, and FS + IS based on the DE method is very effective.

Table 9: Comparison of different feature selection and instance selection methods on OpenOffice.

OpenOffice	Original	IG	CHI	SU	FS
Top-1	0.182	0.0919	0.0919	0.0919	0.1458
Top-2	0.275	0.1563	0.1563	0.1563	0.2155
Top-3	0.3211	0.2123	0.2123	0.2123	0.266
Top-4	0.3507	0.2462	0.2462	0.2462	0.3005
Top-5	0.3775	0.2618	0.2618	0.2618	0.3295
Top-6	0.4008	0.2904	0.2904	0.2904	0.3569
Top-7	0.4209	0.3079	0.3079	0.3079	0.3802
Top-8	0.4437	0.3257	0.3257	0.3257	0.4078
Top-9	0.4636	0.3437	0.3437	0.3437	0.4368
Top-10	0.4839	0.353	0.353	0.353	0.4624
OpenOffice	*Original*	*CNN*	*ENN*	*ICF*	*IS*
Top-1	0.182	0.0289	0.0289	0.0289	**0.1477**
Top-2	0.275	0.0491	0.0491	0.0491	**0.2092**
Top-3	0.3211	0.0735	0.0735	0.0735	**0.2447**
Top-4	0.3507	0.1017	0.1017	0.1017	**0.2723**
Top-5	0.3775	0.1266	0.1266	0.1266	**0.2971**
Top-6	0.4008	0.1479	0.1479	0.1479	**0.3202**
Top-7	0.4209	0.1673	0.1673	0.1673	**0.3844**
Top-8	0.4437	0.1815	0.1815	0.1815	**0.4599**
Top-9	0.4636	0.1992	0.1992	0.1992	**0.5088**
Top-10	0.4839	0.2197	0.2197	0.2197	**0.5376**

Table 10: Comparison of different feature selection and instance selection methods on GCC.

GCC	Original	IG	CHI	SU	FS
Top-1	0.3464	0.1832	0.1832	0.1832	**0.5115**
Top-2	0.525	0.3109	0.3109	0.3109	**0.6724**
Top-3	0.5786	0.4151	0.4151	0.4151	**0.7677**
Top-4	0.6036	0.4899	0.4899	0.4899	**0.7863**
Top-5	0.6241	0.548	0.548	0.548	**0.799**
Top-6	0.6464	0.5912	0.5912	0.5912	**0.8098**
Top-7	0.6607	0.6281	0.6281	0.6281	**0.8224**
Top-8	0.6705	0.6608	0.6608	0.6608	**0.8302**
Top-9	0.6804	0.6817	0.6817	0.6817	**0.8403**
Top-10	0.6884	0.7085	0.7085	0.7085	**0.8451**
GCC	*Original*	*CNN*	*ENN*	*ICF*	*IS*
Top-1	0.3464	0.1832	0.0923	0.1906	0.519
Top-2	0.525	0.3109	0.1869	0.3433	0.6798
Top-3	0.5786	0.4151	0.2695	0.4564	0.7677
Top-4	0.6036	0.4899	0.3325	0.5398	0.7822
Top-5	0.6241	0.548	0.3924	0.5961	0.7949
Top-6	0.6464	0.5912	0.4464	0.6448	0.8016
Top-7	0.6607	0.6281	0.4989	0.6839	0.8179
Top-8	0.6705	0.6608	0.5391	0.7118	0.8232
Top-9	0.6804	0.6817	0.5786	0.7375	0.8306
Top-10	0.6884	0.7085	0.6158	0.7587	0.8369

4.3.5. RQ5: How Can the Developer's Activities Affect the Optimal List of Recommended Developers? In this part, we add the developer engagement level into our experiment to calculate the accuracy on four bug repositories (Mozilla, NetBeans, OpenOffice, and GCC) from Top-1 to Top-10. The results are shown in Tables 19–22 (in Tables 19–22, we use "Without developer engagement" to describe not considering the developer engagement, and "With developer engagement" refers to the consideration of developer engagement).

According to the results, NBM's accuracy improves higher on all feature extraction schemes except for Mozilla. The highest classification accuracies on the four bug repositories have changed compared with accuracies without considering developer engagement. The improved percent of FS + IS can reach up to 2.63% on Mozilla. For NetBeans, the overall accuracies of FS, IS, and FS + IS have substantially improved. The Top-10 accuracies of FS, IS, and FS + IS have gone up 5.64%, 2.47%, and 2.91%, respectively, which verifies our thinking in RQ2. The results indicate that the accuracy of NetBeans with consideration of developer engagement has substantial improvement, which explains that the developer flow is frequent in NetBeans. Table 21 shows that the accuracies of the three proposed methods improve on OpenOffice, and FS has a good improved range of 9.33%. Similarly, considering developer engagement, the

Table 11: Comparison experiment using FS, IS, and FS + IS on Mozilla.

Mozilla	Original	FS -> IS (IG + ICF)	IS -> FS (ICF + IG)	FS	IS	FS + IS
Top-1	0.0906	0.0059	0.0654	0.0868	**0.1208**	0.1001
Top-2	0.1463	0.01	0.0879	0.13	**0.1884**	0.1488
Top-3	0.1833	0.0135	0.1083	0.1578	**0.2105**	0.1756
Top-4	0.2045	0.0197	0.1602	0.178	**0.2351**	0.2072
Top-5	0.2233	0.0214	0.1736	0.1956	**0.2528**	0.2388
Top-6	0.2404	0.0218	0.1916	0.2104	**0.2639**	0.2505
Top-7	0.2551	0.0256	0.1985	0.2287	**0.276**	0.2645
Top-8	0.2669	0.0284	0.21	0.2424	**0.2843**	0.2778
Top-9	0.278	0.0301	0.2269	0.2518	**0.2933**	0.2896
Top-10	0.2889	0.0308	0.2383	0.2601	**0.3044**	0.2944

Table 12: Comparison experiment using FS, IS, and FS + IS on NetBeans.

NetBeans	Original	FS -> IS (IG + ICF)	IS -> FS (ICF + IG)	FS	IS	FS + IS
Top-1	0.2917	0.128	0.1148	0.2532	0.2945	**0.261**
Top-2	0.3972	0.1847	0.1827	0.3524	0.3951	**0.3402**
Top-3	0.4311	0.2279	0.2458	0.4012	0.4529	**0.3974**
Top-4	0.4612	0.2491	0.2786	0.4329	0.4708	**0.4429**
Top-5	0.4847	0.2687	0.3066	0.4576	0.4842	**0.475**
Top-6	0.5091	0.2864	0.3348	0.4732	0.4923	**0.4946**
Top-7	0.5273	0.3015	0.3532	0.4911	0.5019	**0.5133**
Top-8	0.5447	0.314	0.3683	0.5072	0.5176	**0.5325**
Top-9	0.5627	0.3292	0.3819	0.5251	0.5297	**0.5426**
Top-10	0.5759	0.3421	0.3965	0.5414	0.5442	**0.5607**

Table 13: Comparison experiment using FS, IS, and FS + IS on OpenOffice.

OpenOffice	Original	FS -> IS (IG + ICF)	IS -> FS (ICF + IG)	FS	IS	FS + IS
Top-1	0.182	0.0084	0.0313	0.1458	**0.1477**	0.1396
Top-2	0.275	0.0244	0.075	0.2155	**0.2092**	0.2022
Top-3	0.3211	0.0311	0.1061	0.266	**0.2447**	0.2516
Top-4	0.3507	0.0362	0.1554	0.3005	**0.2723**	0.2843
Top-5	0.3775	0.0411	0.1796	0.3295	**0.2971**	0.3122
Top-6	0.4008	0.0642	0.2025	0.3569	**0.3202**	0.3431
Top-7	0.4209	0.0708	0.4196	0.3802	**0.3844**	0.3681
Top-8	0.4437	0.075	0.4427	0.4078	**0.4599**	0.3955
Top-9	0.4636	0.0806	0.4547	0.4368	**0.5088**	0.424
Top-10	0.4839	0.0819	0.4603	0.4624	**0.5376**	0.454

Table 14: Comparison experiment using FS, IS, and FS + IS on GCC.

GCC	Original	FS -> IS (IG + ICF)	IS -> FS (ICF + IG)	FS	IS	FS + IS
Top-1	0.3464	0.2878	0.3414	**0.5115**	0.519	0.4749
Top-2	0.525	0.4494	0.4758	**0.6724**	0.6798	0.6356
Top-3	0.5786	0.5551	0.5566	**0.7677**	0.7677	0.7648
Top-4	0.6036	0.6288	0.6095	**0.7863**	0.7822	0.7884
Top-5	0.6241	0.6742	0.6504	**0.799**	0.7949	0.7989
Top-6	0.6464	0.7096	0.6768	**0.8098**	0.8016	0.8112
Top-7	0.6607	0.736	0.6992	**0.8224**	0.8179	0.8213
Top-8	0.6705	0.7543	0.7148	**0.8302**	0.8232	0.8281
Top-9	0.6804	0.7755	0.736	**0.8403**	0.8306	0.8345
Top-10	0.6884	0.7893	0.7528	**0.8451**	0.8369	0.8393

accuracy of GCC is going up slightly. FS + IS has the biggest improvement of 1.05% among the three proposed approaches.

Meanwhile, the study of developer engagement alleviates the overfitting problem effectively for feature extraction on NetBeans. Moreover, the accuracy of NBM is significantly

Table 15: Reduced data size comparison of FS, IS and FS + IS on Mozilla.

Mozilla	FS + IS	FS	IS	Original information (7 + 1)
IS number	5272	11291	4679	15035
FS number	3528	1163	14686	31317
Data reduction degree of IS	0.3506	0.7510	0.3112	
Data reduction degree of FS	0.1127	0.0371	0.4689	
Overall reduction accuracy (row ∗ column)	0.0395	0.0279	0.1459	

Table 16: Reduced data size comparison of FS, IS, and FS + IS on NetBeans.

NetBeans	FS + IS	FS	IS	Original information (7 + 1)
IS number	16919	16941	5944	17231
FS number	9041	9964	24324	34413
Data reduction degree of IS	0.9819	0.9832	0.3450	
Data reduction degree of FS	0.2627	0.2895	0.7068	
Overall reduction degree (row ∗ column)	0.2580	0.2847	0.2438	

Table 17: Reduced data size comparison of FS, IS, and FS + IS on OpenOffice.

OpenOffice	FS + IS	FS	IS	Original information (7 + 1)
IS number	12527	17941	18001	18781
FS number	1934	2349	24175	44248
Data reduction degree of IS	0.6670	0.9553	0.9585	
Data reduction degree of FS	0.0437	0.0531	0.5464	
Overall reduction degree (row ∗ column)	0.0292	0.0507	0.5237	

Table 18: Reduced data size comparison of FS, IS, and FS + IS on GCC.

GCC	FS + IS	FS	IS	Original information (7 + 1)
IS number	I7493	10742	6394	11169
FS number	3842	10468	26170	46251
Data reduction degree of IS	0.6709	0.9618	0.5725	
Data reduction degree of FS	0.0831	0.2263	0.5658	
Overall reduction degree (row ∗ column)	0.0557	0.2177	0.3239	

Table 19: Accuracy of Top-k considering developer engagement and without considering engagement on Mozilla.

Mozilla	Without developer engagement	With developer engagement
FS		
Top-1	0.0868	0.0868
Top-2	0.13	0.13
Top-3	0.1578	0.1578
Top-4	0.178	0.178
Top-5	0.1956	0.1956
Top-6	0.2104	0.2104
Top-7	0.2287	0.2287
Top-8	0.2424	0.2424
Top-9	0.2518	0.2518
Top-10	0.2601	0.2601
IS		
Top-1	0.1208	**0.1236**
Top-2	0.1884	**0.1911**
Top-3	0.2105	**0.2123**
Top-4	0.2351	**0.2382**
Top-5	0.2528	**0.2548**
Top-6	0.2639	**0.2715**
Top-7	0.276	**0.2777**
Top-8	0.2843	**0.2881**
Top-9	0.2933	**0.3012**

Table 19: Continued.

Mozilla	Without developer engagement	With developer engagement
Top-10	0.3044	**0.3096**
FS + IS		
Top-1	0.1001	**0.1071**
Top-2	0.1488	**0.159**
Top-3	0.1756	**0.204**
Top-4	0.2072	**0.2398**
Top-5	0.2388	**0.2607**
Top-6	0.2505	**0.2778**
Top-7	0.2645	**0.2853**
Top-8	0.2778	**0.3003**
Top-9	0.2896	**0.3116**
Top-10	0.2944	**0.3207**

Table 20: Accuracy of Top-k considering engagement and without considering engagement on NetBeans.

NetBeans	Without developer engagement	With developer engagement
FS		
Top-1	0.2532	**0.2521**
Top-2	0.3524	**0.3578**
Top-3	0.4012	**0.4173**
Top-4	0.4329	**0.4496**
Top-5	0.4576	**0.4734**
Top-6	0.4732	**0.4919**
Top-7	0.4911	**0.523**
Top-8	0.5072	**0.5518**
Top-9	0.5251	**0.5815**
Top-10	0.5414	**0.6342**
IS		
Top-1	0.2945	**0.2995**
Top-2	0.3951	**0.3972**
Top-3	0.4529	**0.4423**
Top-4	0.4708	**0.4671**
Top-5	0.4842	**0.4824**
Top-6	0.4923	**0.4937**
Top-7	0.5019	**0.51**
Top-8	0.5176	**0.5277**
Top-9	0.5297	**0.5544**
Top-10	0.5442	**0.628**
FS + IS		
Top-1	0.261	**0.2427**
Top-2	0.3402	**0.3551**
Top-3	0.3974	**0.4073**
Top-4	0.4429	**0.4406**
Top-5	0.475	**0.4649**
Top-6	0.4946	**0.4876**
Top-7	0.5133	**0.5197**
Top-8	0.5325	**0.5452**
Top-9	0.5426	**0.5717**
Top-10	0.5607	**0.6246**

higher than the original accuracy. Because the accuracy of the feature extraction scheme of FS + IS has been obviously improved, we conclude that the introduction of developer engagement can successfully compensate for the problem of information loss caused by FS + IS.

We find that the accuracy on the test set using the NBM classifier generally presents a trend of increasing first, then lowering, and finally, flattening with increasing N. After a careful analysis, we find that the two kinds of information compete with the growth of N. One is the effective information related to the test set, and the other is the noise. This confrontation relationship leads to the accuracy increasing first and then decreasing with the growth of N. For the data reduction of Mozilla_total's DE_FS, the accuracy is always at a lower degree with N changing. It is because the flow of Mozilla_total staff changes frequently with time, which causes the noisy and redundant data to grow and fragment. However, we also found that the dataset generated by

Table 21: Accuracy of Top-k considering engagement and without considering engagement on OpenOffice.

OpenOffice	Without developer engagement	With developer engagement
FS		
Top-1	0.1458	**0.1703**
Top-2	0.2155	**0.2409**
Top-3	0.266	**0.2861**
Top-4	0.3005	**0.3208**
Top-5	0.3295	**0.3562**
Top-6	0.3569	**0.3951**
Top-7	0.3802	**0.4357**
Top-8	0.4078	**0.4889**
Top-9	0.4368	**0.5256**
Top-10	0.4624	**0.5557**
IS		
Top-1	0.1477	**0.1646**
Top-2	0.2092	**0.2247**
Top-3	0.2447	**0.2636**
Top-4	0.2723	**0.3002**
Top-5	0.2971	**0.3724**
Top-6	0.3202	**0.455**
Top-7	0.3844	**0.5059**
Top-8	0.4599	**0.5392**
Top-9	0.5088	**0.5574**
Top-10	0.5376	**0.5741**
FS + IS		
Top-1	0.1396	**0.1579**
Top-2	0.2022	**0.2305**
Top-3	0.2516	**0.2717**
Top-4	0.2843	**0.3106**
Top-5	0.3122	**0.3434**
Top-6	0.3431	**0.381**
Top-7	0.3681	**0.4197**
Top-8	0.3955	**0.4656**
Top-9	0.424	**0.5023**
Top-10	0.454	**0.5315**

Table 22: Accuracy of Top-k considering engagement and without considering engagement on GCC.

GCC	Without developer engagement	With developer engagement
FS		
Top-1	0.5115	**0.5115**
Top-2	0.6724	**0.6724**
Top-3	0.7677	**0.7673**
Top-4	0.7863	**0.7863**
Top-5	0.799	**0.7986**
Top-6	0.8098	**0.8094**
Top-7	0.8224	**0.8235**
Top-8	0.8302	**0.8347**
Top-9	0.8403	**0.8436**
Top-10	0.8451	**0.8507**
IS		
Top-1	0.519	**0.519**
Top-2	0.6798	**0.6798**
Top-3	0.7677	**0.7681**
Top-4	0.7822	**0.7807**
Top-5	0.7949	**0.7937**
Top-6	0.8016	**0.8116**
Top-7	0.8179	**0.8213**
Top-8	0.8232	**0.8284**
Top-9	0.8306	**0.838**
Top-10	0.8369	**0.847**

Table 22: Continued.

GCC	Without developer engagement	With developer engagement
FS + IS		
Top-1	0.4749	**0.4753**
Top-2	0.6356	**0.6352**
Top-3	0.7648	**0.7652**
Top-4	0.7884	**0.788**
Top-5	0.7989	**0.7996**
Top-6	0.8112	**0.8131**
Top-7	0.8213	**0.8213**
Top-8	0.8281	**0.8303**
Top-9	0.8345	**0.8382**
Top-10	0.8393	**0.8498**

Table 23: Distance corresponding to the peak of the engagement accuracy.

Bug repositories	FS + IS	Total number	FS	Total number	IS	Total number
GCC	231	7493	1095	10742	195	6394
OpenOffice	162	12527	176	17941	288	18001
NetBeans	502	5272	11291	11291	596	4679
Mozilla	464	16941	464	16941	460	5944

Mozilla_total's DE_IS scheme performs well after adding developer engagement, which indicates that the denoising ability of IS on the Mozilla_total is better than that of the FS approach.

We conclude that the introduction of developer engagement can effectively improve the classification accuracy of NBM. Moreover, it significantly alleviates the overfitting phenomenon which happens when a model learns the details and noise in the training data to the extent that it negatively impacts the performance of the model on unseen data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact NBM's ability to generalize. In our improved NBM classification, removal of redundant features from bug trial dataset can prevent overfitting. In addition, the developer engagement effectively compensates for information loss caused by the FS + IS method and substantially increases the accuracy. Meanwhile, compared with the overall dataset, the optimal reference range is smaller and easier to implement if considering developer engagement.

In Table 23, we analyze the peak of engagement, which can explain the frequency of developers' flow in different bug repositories. A greater distance between the peak and the average means a higher frequency of recent personnel movements. We can learn that the developers of Mozilla and OpenOffice flow more frequently than GCC and NetBeans.

5. Conclusion and Future Work

In this paper, we propose a new bug triage method for recommending suitable developers to fix newly reported bugs. To solve the problem of small search range and neglecting the chronological order in the traditional bug triage method, we improve the existing heuristic search method and expand the search scope further based on the chronological order of bug reports. We find that developer engagement has an impact on bug triage; therefore, in addition to the text information provided in the bug report, we consider the developer's product information to recommend the best developer for the new bug report. We use FS, IS, and FS + IS to verify our approach on four bug repositories: GCC, OpenOffice, Mozilla, and NetBeans. The results show that the method proposed in this paper is more effective than the previous methods. In future work, we plan to verify our approach using more bug repositories. Moreover, we plan to apply our method to additional software projects.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (nos. 61672122, 61602077, 61771087, 51879027, 51579024, 71831002, and 61902050), Program for Innovative Research Team in University of Ministry of Education of China (no. IRT 17R13), the Fundamental Research Funds for the Central Universities (nos. 3132019501 and 3132019502), and the Next-Generation Internet Innovation Project of CERNET (nos. NGII20181203, NGII20181205, and NGII201810128).

References

[1] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What makes a good bug report?"

IEEE Transactions on Software Engineering, vol. 36, no. 5, pp. 618–643, 2010.

[2] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th International Conference on Software Engineering*, pp. 361–370, Shanghai, China, May 2006.

[3] W. Deng, H. Zhao, L. Zou, G. Li, X. Yang, and D. Wu, "A novel collaborative optimization algorithm in solving complex optimization problems," *Soft Computing*, vol. 21, no. 15, pp. 4387–4398, 2017.

[4] S. Guo, R. Chen, H. Li, T. Zhang, and Y. Liu, "Identify severity bug report with distribution imbalance by CR-SMOTE and ELM," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 2, pp. 139–175, 2019.

[5] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage," *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 3, pp. 1–35, 2011.

[6] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *Proceedings of the 2012 34th International Conference on Software Engineering (ICSE)*, IEEE, Zurich, Switzerland, pp. 25–35, June 2012.

[7] H. Li, G. Gao, R. Chen, X. Ge, S. Guo, and L.-Y. Hao, "The influence ranking for testers in bug tracking systems," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 1, pp. 93–113, 2019.

[8] W. Pan and C. Chai, "Structure-aware mashup service clustering for cloud-based internet of things using genetic algorithm based clustering algorithm," *Future Generation Computer Systems*, vol. 87, pp. 267–277, 2018.

[9] W. Pan, J. Dong, K. Liu, and J. Wang, "Topology and topic-aware service clustering," *International Journal of Web Services Research*, vol. 15, no. 3, pp. 18–37, 2018.

[10] https://www.mantisbt.org/.

[11] https://www.bugzilla.org/.

[12] https://www.atlassian.com/software/jira.

[13] S. Guo, Y. Liu, R. Chen, X. Sun, and X. Wang, "Improved SMOTE algorithm to deal with imbalanced activity classes in smart homes," *Neural Processing Letters*, vol. 50, no. 2, pp. 1503–1526, 2019.

[14] S. Guo, R. Chen, M. Wei, H. Li, and Y. Liu, "Ensemble data reduction techniques and multi-RSMOTE via fuzzy integral for bug report classification," *IEEE Access*, vol. 6, pp. 45934–45950, 2018.

[15] W. Deng, J. Xu, and H. Zhao, "An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem," *IEEE Access*, vol. 7, pp. 20281–20292, 2019.

[16] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer, "Categorizing bugs with social networks: a case study on four open source software communities," in *Proceedings of the 2013 35th International Conference on Software Engineering (ICSE)*, IEEE, San Francisco, CA, USA, pp. 1032–1041, 2013.

[17] H. Hu, H. Zhang, J. Xuan, and W. Sun, "Effective bug triage based on historical bug-fix information," in *Proceedings of the 2014 IEEE 25th International Symposium on Software Reliability Engineering*, IEEE, Naples, Italy, pp. 122–132, November 2014.

[18] W. Pan, B. Song, K. Li, and K. Zhang, "Identifying key classes in object-oriented software using generalizedk-core decomposition," *Future Generation Computer Systems*, vol. 81, pp. 188–202, 2018.

[19] H. Jiang, L. Nie, Z. Sun et al., "ROSF: leveraging information retrieval and supervised learning for recommending code snippets," *IEEE Transactions on Services Computing*, vol. 12, no. 1, pp. 34–46, 2019.

[20] R. Chen, S.-K. Guo, X.-Z. Wang, and T.-L. Zhang, "Fusion of multi-RSMOTE with fuzzy integral to classify bug reports with an imbalanced distribution," *IEEE Transactions on Fuzzy Systems*, vol. 27, no. 12, pp. 2406–2420, 2019.

[21] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 111–120, New York, NY, USA, August 2009.

[22] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," in *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, IEEE, Washington, DC, USA, pp. 1–10, 2010.

[23] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, "Fuzzy set and cache-based approach for bug triaging," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, pp. 365–375, New York, NY, USA, 2011.

[24] W. Pan, B. Li, J. Liu, Y. Ma, and B. Hu, "Analyzing the structure of Java software systems by weightedK-core decomposition," *Future Generation Computer Systems*, vol. 83, pp. 431–444, 2018.

[25] K. Somasundaram and G. C. Murphy, "Automatic categorization of bug reports using latent dirichlet allocation," in *Proceedings of the 5th India Software Engineering Conference*, pp. 125–130, Kanpur, India, February 2012.

[26] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang, "Improving automated bug triaging with specialized topic model," *IEEE Transactions on Software Engineering*, vol. 43, no. 3, pp. 272–297, 2016.

[27] G. Yang, T. Zhang, and B. Lee, "Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports," in *Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference*, IEEE, Vasteras, Sweden, pp. 97–106, July 2014.

[28] W. Deng, S. Zhang, H. Zhao, and X. Yang, "A novel fault diagnosis method based on integrating empirical wavelet transform and fuzzy entropy for motor bearing," *IEEE Access*, vol. 6, pp. 35042–35056, 2018.

[29] W. Deng, H. Zhao, X. Yang, J. Xiong, M. Sun, and B. Li, "Study on an improved adaptive PSO algorithm for solving multi-objective gate assignment," *Applied Soft Computing*, vol. 59, pp. 288–302, 2017.

[30] H. Jiang, X. Li, Z. Ren, J. Xuan, and Z. Jin, "Toward better summarizing bug reports with crowdsourcing elicited attributes," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 2–22, 2019.

[31] H. Zhao, M. Sun, W. Deng, and X. Yang, "A new feature extraction method based on EEMD and multi-scale fuzzy entropy for motor bearing," *Entropy*, vol. 19, no. 1, p. 14, 2016.

[32] J. Xuan, H. Jiang, Y. Hu et al., "Towards effective bug triage with software data reduction techniques," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 1, pp. 264–280, 2015.

[33] H. Zhao, R. Yao, L. Xu, Y. Yuan, G. Li, and W. Deng, "Study on a novel fault damage degree identification method using high-order differential mathematical morphology gradient spectrum entropy," *Entropy*, vol. 20, no. 9, p. 682, 2018.

[34] Y. Xiang, W. Pan, H. Jiang, Y. Zhu, and H. Li, "Measuring software modularity based on software networks," *Entropy*, vol. 21, no. 4, p. 344, 2019.

[35] H. Naguib, N. Narayan, B. Brügge, and D. Helal, "Bug report assignee recommendation using activity profiles," in *Proceedings of the 2013 10th Working Conference on Mining Software Repositories (MSR)*, IEEE, San Francisco, CA, USA, pp. 22–30, May 2013.

[36] T. Zhang, G. Yang, B. Lee, and E. K. Lua, "A novel developer ranking algorithm for automatic bug triage using topic model and developer relations," in *Proceedings of the 2014 21st Asia-Pacific Software Engineering Conference*, pp. 223–230, Jeju, Republic of Korea, December 2014.

[37] X. Xia, D. Lo, X. Wang, and B. Zhou, "Accurate developer recommendation for bug resolution," in *Proceedings of the 2013 20th Working Conference on Reverse Engineering (WCRE)*, IEEE, Koblenz, Germany, pp. 72–81, October 2013.

[38] M. Linares-Vásquez, K. Hossen, H. Dang, H. Kagdi, M. Gethers, and D. Poshyvanyk, "Triaging incoming change requests: bug or commit history, or code authorship?" in *Proceedings of the 2012 28th IEEE International Conference on Software Maintenance (ICSM)*, IEEE, Trento, Italy, pp. 451–460, September 2012.

[39] K. Kevic, S. C. Müller, T. Fritz, and H. C. Gall, "Collaborative bug triaging using textual similarities and change set analysis," in *Proceedings of the 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, IEEE, San Francisco, CA, USA, pp. 17–24, May 2013.

[40] S. Wang and D. Lo, "Version history, similar report, and structure: putting them together for improved bug localization," in *Proceedings of the 22nd International Conference on Program Comprehension*, pp. 53–63, Hyderabad, India, June 2014.

[41] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation," in *Proceedings of the 2013 10th Working Conference on Mining Software Repositories (MSR)*, IEEE, San Francisco, CA, USA, pp. 2–11, May 2013.

[42] T. Zhang and B. Lee, "An automated bug triage approach: a concept profile and social network based developer recommendation," in *Proceedings of the International Conference on Intelligent Computing*, pp. 505–512, Huangshan, China, July 2012.

[43] D. Challet and A. Lombardoni, "Bug propagation and debugging in asymmetric software structures," *Physical Review E*, vol. 70, no. 4, Article ID 046109, 2004.

[44] W.-F. Pan, B. Li, Y.-T. Ma, Y.-Y. Qin, and X.-Y. Zhou, "Measuring structural quality of object-oriented softwares via bug propagation analysis on weighted software networks," *Journal of Computer Science and Technology*, vol. 25, no. 6, pp. 1202–1213, 2010.