IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received August 4, 2018, accepted August 31, 2018, date of publication September 10, 2018, date of current version October 12, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2869306

Rapid and Efficient Bug Assignment Using ELM for IOT Software

YING YIN[1], (Member, IEEE), XIANGJUN DONG[2], AND TIANTIAN XU[2]

[1]College of Computer Science and Engineering, Northeastern University, Shenyang 110819, China
[2]School of Information, Qilu University of Technology (Shandong Academy of Sciences), Jinan 250300, China

Corresponding authors: Ying Yin (yinying@mail.neu.edu.cn) and Tiantian Xu (xtt-ok@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61772124 and Grant 61332014 and in part by the Fundamental Research Funds for the Central Universities under Grant N150402002.

ABSTRACT The reliable implementation of software in an Internet system directly influences information transmission especially for the Internet of Things (IoT) system. Once defects in the system are found, the communication between things and things and the interaction between people and things in the IoT will be greatly affected. Therefore, rapid and effective defect assignment to the right developer is the key to ensuring software quality and bringing down time consumption in the IoT software life cycle. However, as the size of the software becomes increasing larger, the requirement of users grows, and a large number of software bugs will be found every day. It is difficult for managers to assign the software defects to the appropriate developers. In this paper, a novel hybrid method based on a diversified feature selection and an ensemble extreme learning machine (ELM) is proposed. First, the useful information is extracted from defect reports; then, the data are preprocessed to establish a vector space model; and the diversified feature selection is preprocessed in order to select a smallest set of representative non-redundant features with maximal statistical information. Finally, an ensemble GA-based ELM training classifier is used. Experimental results show that, compared to SVM, C4.5, NaiveBayes, and KNN classifiers, the proposed ELM-based bug triage approach with representative feature selection techniques in this paper significantly improves the efficiency and the effectiveness of bug triages.

INDEX TERMS IoT software, diversified feature selection, software defect triage, ensemble extreme learning machine (ELM).

I. INTRODUCTION

The Internet of Things industry is developing rapidly all over the world. The development of the Internet of Things industry is dependent on the popularization of software products. Software covers many aspects of the Internet of Things system, such as the IOT terminal system, short-distance connections, IOT architecture, IOT browser, IOT security and other important applications [3]. According to an OMA survey, 60% of IOT enterprises believe that IOT products rely on open source software. According to the Vision Mobile survey, 91% of Internet of Things developers have used open bug repositories to seek help to repair defects and further improve the performance and safety of Internet of Things software. Reliable quality of software is the basis of ensuring the efficient and safe operation of the IOT system.

Open software projects usually adopt an open bug repository, such as Bugzilla or JIRA to manage software defect reports that were submitted by developers and users from the process of software development and maintenance [1]. When one or some software bugs are found, software developers and users will submit bug reports to the repository in the form of software defect reports. A bug report describes the details or the features (such as version, description, component, and severity) of the submitted bug [2]. The advantage of the open repository is that it allows more bug reports to be submitted, confirmed and assigned.

Traditional artificial defect triage is described below: First, test group members found the software defects and generated defect reports and then handed in the reports to the test group leader. Further, according to the defect reports, test group leader first judged whether or not it is a genuine bug report and determined the validity of the software defects, and then, the report is checked for duplication. If the reports are effective, then the numbers hand it over to the team leader. Finally, the team leader assigns the software bug reports to the most suitable developers according to their experience.

Therefore, the project manager can use the system to assign the reports, manage the solution process and improve the quality of the software project. However, bug assignment expenses incur significant costs, such as time and human resources, especially for rapidly growing projects with many bug reports generated every day. For instance, there are on average more than 29 reports submitted to the platform each day in Eclipse [18]. Up to May 1, 2014, data show that Bugzilla has managed over 434,000 and 1,004,000 bug reports for Eclipse and Mozilla, respectively. Each bug report must be determined, validated and assigned to an appropriate developer based on its features(whether it is a new problem or an enhancement).

The goal of most previous work on bug triage is to analyze bug reports to help developers or programmers repair the bugs efficiently. Jeong *et al*. put forward a graph model based on Markov chains to detect the defect information modified. Cubranic *et al.* [35] proposed a method that uses the Naive Bayes algorithm to convert the defect triage into text categorization. Since then, defect assignments based on text classification technology have been used. Nevertheless, the accuracies of the above two methods must to be improved when using these methods to triage defects. Xuan *et al*. [33] improved the accuracy of the classification by reducing the size of data using instances and selected feature combinations, improve the quality of the data, and selecting the appropriate instance selection and feature selection order. Ahsan *et al.* [34] mapped the defect report information to the latent semantic space, reduced the size of the data using singular value decomposition, improved the quality of the data, and then improved the accuracy of the defect assignments. Other studies use machine learning-based bug report triage methods to recommend suitable repairers [30], [32]. The method must provide a bug report with a repairer, extract some features of the existing bug report, train a model through machine learning, and then execute predictions on unknown bug reports; Other classification techniques, such as SVM, naive Bayes and decision trees are also used for triage [29], [31], [32]. In these methods, a vector space model is used to digitize bug reports. First, find the useless words in the text, select the other words, retain the more important words and calculate their weights. Defect assignment papers based on association rules have been presented in the literature. However, the large number of rules affects the efficiency of assignments. All in all, there are two major challenges to existing methods in the triage of defects: (1) The speed of bug triage should be improved; with the advent of the Internet era, increasingly more platforms support a variety of software, such as mobile phones, IPads and various mobile terminals; and in the face of large-scale bug reports, it is necessary to improve the efficiency and accuracy of bug allocation. (2) The triage accuracy must be improved; existing bug assignment methods for single report feature extraction and assignment accuracy must be improved. With the increasing size and complexity of software projects, an increasingly more bugs are generated. When a bug report is generated, it is important to assign it quickly and effectively to the appropriate repairer. Therefore, we require a fast and accurate defect triage algorithm to reduce the consumption of time and labor. Below are the primary contributions of this paper:

In this paper, we propose an effective bug report feature learning model and an efficient bug assignment mechanism for the above challenges. The primary contributions of this article are as follows: (1) we propose a new hybrid bug triage method based on supervised learning and ensemble ELM classification based on GA. (2) We propose an efficient method to identify diversified features as the representative feature subsets of an all-dimensional space for each bug report. Not using all features to construct a large searching space, we only select those representative features with high relevance and diversification by adopting a greedy algorithm; and (3) We design an optimal ensemble ELM strategy by combining the selected diversified features to promote the strategy's effectiveness.

The remainder of this paper is organized as follows. Section 2 gives the background of bug triage, including the information extracted from a bug report and the life-cycle of a bug report. Section 3 presents the theory of the extreme learning machine. Section 4 presents the hybrid ELM-based bug triage framework. Section 5 describes the representative feature selection and gives the process of solution in detail. Experimental analysis is elaborated in Section 6. Section 7 describes the previous related work on bug triage. Finally, Section 7 concludes this paper.

II. BUG REPORT

Currently, bug repositories, such as Bugzilla, GNAT and JIRA are widely used in software projects to collect bug information, store bug reports and triage the bug fixers. Bug repositories play a very important role because they provide a communication channel for software developers around the world. With these repositories, many bug reports can be repaired in time. The emergence of an increasing number of bug repositories effectively improves the quality of software development. Bugs in repositories takes many forms, such as new features, defects, update operations or refactoring. Once a new software bug is found, this bug will be recorded by a recorder and submitted to the bug repository. The bug repository provides a database of problems for a software project. There are a series of states for a bug report over its lifecycle. We elaborate these states as a lifecycle graph of a bug report. Figure 1 shows the several states of a bug project for internet software. Other projects have similarities to this model.

At first, the state of a bug report is marked as NEW when it is submitted to the bug repository. For a fixer/developer, once he/she has been assigned to the report, the status of bug report becomes ASSIGNED and the fixer/developer becomes an assignee. Once the report is closed, the status of a bug report is set to RESOLVED. It may further become VERIFIED or CLOSED when it is reopened again. The bug report provides an integrated record of a bug from being presented to

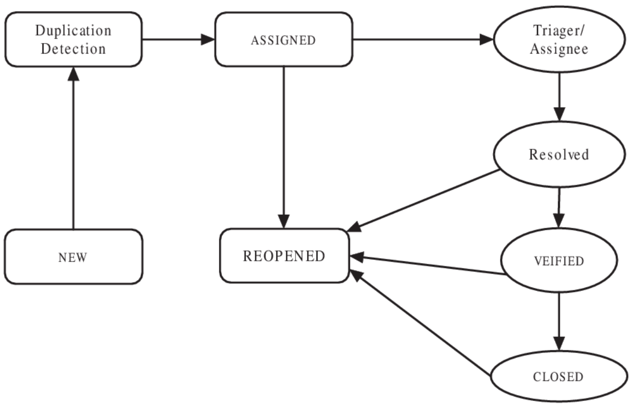
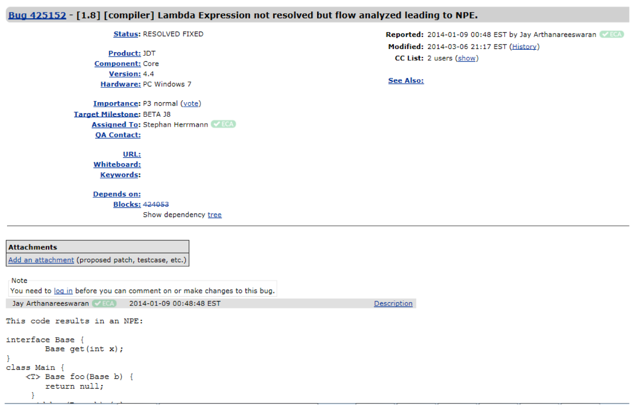


FIGURE 1. The lifecycle of a bug report in Bugzilla.

being repaired or closed. Its content mainly includes (1) the migration process of the reported bug, such as new, assigned, reopened, resolved, closed, or fixed, and (2) the repair suggestions of the reviewer and the comments as well as the time of posting.

Bug reports are similar in form in the same bug software repository, whether it be IOT software or network software. We take the Eclipse project as an example; Figure 2 shows a partial bug report on bug 425152 in Eclipse stored in Bugzilla. Each bug report consists of four parts: freeform text, predefined fields, attachments, and dependencies. Some items, such as such as the report creation date, identification number and reporter are provided when the report is created. Other attributes, such as the status (e.g., resolved or fixed), product (e.g., JDT), component (e.g., Core), hardware (e.g., PC Windows 7), version (e.g., 4.4), importance (e.g., P3), and target milestone (e.g., BETA J8), are described by the reporter when the report is submitted, but the report's status may be changed with modifications by the contributor. Figure 2 shows that a contributor named Jay Arthanareeswaran introduced a bug report on Jan.9, 2014. Jay Arthanareeswaran gave a detailed description of this bug about "[compiler] Lambda Expression not resolved but flow analyzed leading to NPE". In addition, he described some



Bug 425152 - [1.8] [compiler] Lambda Expression not resolved but flow analyzed leading to NPE.

Status: RESOLVED FIXED

Product: JDT
Component: Core
Version: 4.4
Hardware: PC Windows 7

Importance: P3 normal (vote)
Target Milestone: BETA J8
Assigned To: Stephan Herrmann ✓ ECA
QA Contact:

URL:
Whiteboard:
Keywords:

Depends on:
Blocks: ~~424053~~
Show dependency tree

Reported: 2014-01-09 00:48 EST by Jay Arthanareeswaran ✓ ECA
Modified: 2014-03-06 21:17 EST (History)
CC List: 2 users (show)

See Also:

Attachments
Add an attachment (proposed patch, testcase, etc.)

Note
You need to log in before you can comment on or make changes to this bug.

Jay Arthanareeswaran ✓ ECA 2014-01-09 00:48:48 EST Description

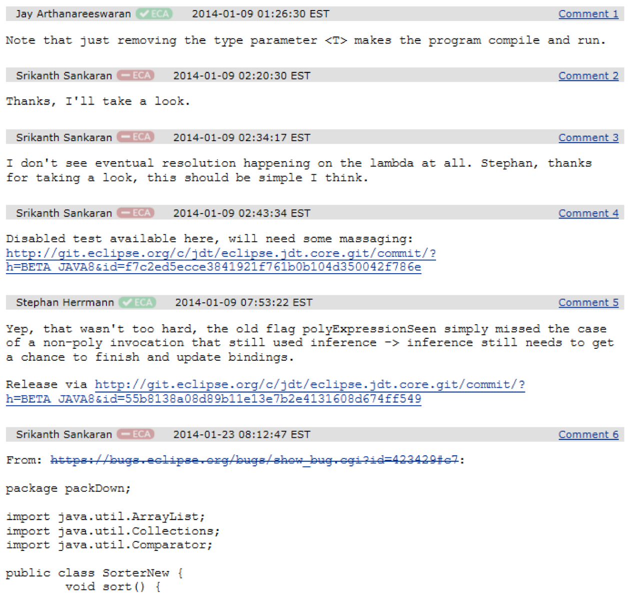
```
This code results in an NPE:

interface Base {
        Base get(int x);
}
class Main {
    <T> Base foo(Base b) {
        return null;
     }
```

FIGURE 2. An example of bug report in Bugzilla.

items associated with this bug report, including components, product and target milestone. Along with the progress of the fix, the status of bug report may change as well.

The summary and description of the bug report is described by the freeform text which consists of a description, a summary and comments. The comments are posted by contributors. These contributors are happy to join in the discussion or have interests in fixing the bug of the project. Generally, contributors can also attach nontextural information, such as test cases and patches. For example, after Jay Arthanareeswaran submitted Bug 425152, there are 9 total comments from 4 contributors, such as Jay Arthanareeswaran, Srikanth Sankaran, Stephan Herrmann and Noopur Gupta. In Figure 3, we only display a part of the comments for bug 425152 in Eclipse stored in Bugzilla. Interestingly, Srikanth Sankaran also contributed 7 comments, Stephan Herrmann contributed 2 comments and Noopur Gupta also contributed 1 comment.



Jay Arthanareeswaran ✓ ECA 2014-01-09 01:26:30 EST Comment 1

```
Note that just removing the type parameter <T> makes the program compile and run.
```

Srikanth Sankaran ECA 2014-01-09 02:20:30 EST Comment 2

```
Thanks, I'll take a look.
```

Srikanth Sankaran ECA 2014-01-09 02:34:17 EST Comment 3

```
I don't see eventual resolution happening on the lambda at all. Stephan, thanks
for taking a look, this should be simple I think.
```

Srikanth Sankaran ECA 2014-01-09 02:43:34 EST Comment 4

```
Disabled test available here, will need some massaging:
http://git.eclipse.org/c/jdt/eclipse.jdt.core.git/commit/?
h=BETA_JAVA8&id=f7c2ed5ecce3841921f761b0b104d350042f786e
```

Stephan Herrmann ✓ ECA 2014-01-09 07:53:22 EST Comment 5

```
Yep, that wasn't too hard, the old flag polyExpressionSeen simply missed the case
of a non-poly invocation that still used inference -> inference still needs to get
a chance to finish and update bindings.

Release via http://git.eclipse.org/c/jdt/eclipse.jdt.core.git/commit/?
h=BETA_JAVA8&id=55b8138a08d89b11e13e7b2e4131608d674ff549
```

Srikanth Sankaran ECA 2014-01-23 08:12:47 EST Comment 6

```
From: https://bugs.eclipse.org/bugs/show_bug.cgi?id=423429#c7:

package packDown;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class SorterNew {
        void sort() {
```

FIGURE 3. The comments of bug 425152 in Bugzilla.

III. INTRODUCTION OF ELM

In recent years, a new artificial neural network model learning method called extreme learning machine (ELM) has been proposed. Extreme learning machine (ELM) is a generalized single hidden-layer feedforward network. ELM described in Algorithm 1 provides better generalization performance with faster speeds than those of traditional feedforward neural network learning models [19]. In ELM, the hidden-layer node parameter is mathematically obtained instead of being iteratively tuned. The learning model not only ensures that the network has good generalization performance, but also greatly promotes the learning speed of the neural network and avoids many problems of using gradient descent learning methods, such as falling into local minima and slow convergence rates.

As an excellent learning model, ELM has been applied in many fields successfully. For example, in [20], ELM was applied for plain text classification by using the OAO strategy and OAA scheme. An ELM-based XML document classification framework was proposed to improve the classification accuracy with an efficient voting strategy [19]. An ELM-based protein secondary prediction framework was proposed [22] to provide excellent performance with high speed. An ELM-based protein-protein interaction prediction framework on multichain sets was proposed by [21]; it uses ELM and SVM as comparisons. Studies indicate that ELM obtains a better recall than of SVM and shows an advantage with its higher speeds.

The work [23] proposes an ELM-based classification on a microarry dataset, evaluating the classification performance on several microarray datasets. The use of ELM for multiresolution access of shadow map compression was proposed in [24]. The ELM-based optimization method for classification was elaborated in [25]. In recent years, the use of intelligent optimization algorithm combined with evolutionary limit learning machine (E-ELM), combined with differential evolution and the ELM algorithm, to improve the performance of SLFNs; In [4], an adaptive DE algorithm is used to optimize the hidden layer node parameters of SLFNs, and Moore-Penrose (MP) generalized inverse acquisition is used to optimize the hidden layer node parameters of SLFNs. IPSO-ELM was presented in [6], optimization of hidden layer node parameters in SLFNs using improved particle swarm Optimization. The output weights of SLFNs can be obtained using ELM, so the output weights of SLFNs can be improved. The performance of SLFNs has been improved. The algorithm has good performance in regression calculation and classification [7].

As mentioned, ELM is a classifier based on SLFN. Standard SLFNs with M arbitrary samples $(\mathbf{x_i}, \mathbf{t_i}) \in \mathbf{R^{m\times n}}$ and activation function $g(x)$ are refer to [16] as

$$\sum_{i=1}^{H} \beta_i g_i(\mathbf{x_j}) = \sum_{i=1}^{H} \beta_i g(\mathbf{w_i} \cdot \mathbf{x_j} + d_i) = \mathbf{o_i}, (j = 1, \ldots, M) \quad (1)$$

where the number of hidden layer nodes is H, the weight vector between the i^{th} hidden node and the input nodes is $\mathbf{w_i} = [w_i^1, w_i^2, \ldots, w_i^n]^T$, the weight vector between the i^{th} hidden node and the output nodes is $\beta_i = [\beta_i^1, \beta_i^2, \ldots, \beta_i^m]^T$, and the threshold of the i^{th} hidden node is b_i. The output of ELM is:

$$f(x) = \sum_{i=1}^{N} \beta_i g(\mathbf{c_i}, d_i, \mathbf{x}) \quad (2)$$

where

$$H(\mathbf{w_1}, \ldots, \mathbf{w_L}, d_1, \ldots, d_L, \mathbf{x_1}, \ldots, \mathbf{x_L}) = \begin{bmatrix} g(\mathbf{w_1} \cdot \mathbf{x_1} + d_1) & \ldots & g(\mathbf{w_L} \cdot \mathbf{x_1} + d_L) \\ \vdots & \ldots & \vdots \\ g(\mathbf{w_1} \cdot \mathbf{x_M} + d_1) & \ldots & g(\mathbf{w_L} \cdot \mathbf{x_M} + d_L) \end{bmatrix}_{M\times L},$$

$$\beta = [\beta_1^T, \ldots, \beta_L^T]_{n\times L}^T$$

The decision function for classification [26] is:

$$d(x) = sign(\sum_{i=1}^{L} \beta_i g(\mathbf{c_i}, d_i, \mathbf{x})) = sign(\beta \cdot \mathbf{H}) \quad (3)$$

When g(x) approximates the M samples with zero error with $\Sigma_{j=1}^{L} \|o_j - t_j\| = 0$ and outputs β_i, w_i and d_i such that

$$\sum_{i=1}^{L} \beta_i g(\mathbf{w_i} \cdot \mathbf{x_j} + d_i) = \mathbf{t_j}, j = 1, \ldots, M \quad (4)$$

The equation above can be expressed compactly as follows:

$$\mathbf{H}\beta = \mathbf{T} \quad (5)$$

where $\mathbf{T} = [\mathbf{t}_1^T, \ldots, \mathbf{t}_L^T]_{m\times L}^T$.

All in all, ELM is relatively rapid compared with the traditional learning algorithms. ELM tends to have not only the smallest training error but also the smallest norm of weights [27]. More detailed introductions to ELM and its improvements can be found in a series of new published literatures.

Algorithm 1 ELM

Input: ODB: original dataset, HLN: Number of Hidden Layer nodes, AF: ActivationFunction

Output: R

```
1. for $k = 1$ to $L$ do
2.     set input weight $w_i$ randomly
3.     set bias $d_i$ randomly
4. calculate $\mathbf{H}$
5. obtain $\beta = \mathbf{H}^\dagger \mathbf{T}$
```

IV. OVERVIEW OF DEFECT TRIAGE

In this section, we first give the ELM-based defect triage framework (Fig.4) and then explain every step of this framework in the following steps respectively. Assume an original dataset of s bug reports, m features/variables and k workers, i.e. S = $\{s_1, s_2, \ldots, s_n\}$, M = $\{f_1, f_2, \ldots, f_m\}$ and C = $\{c_1, c_2, \ldots, c_k\}$. Each sample s_i can be represented by a vector $s_i = \{x_{i1}, x_{i2}, \ldots, x_{im}\}$, where x_{ij} denotes the values of sample s_i on feature f_j. Each sample $s_i \in S$ corresponds to a class label c_i. As shown in Figure 4, the defect triage framework consists of three major steps: Software defect information extraction, representative feature space construction, and optimal ensemble ELM classification based on GA and recommendation.

In order to understand the proposed method better, we first give a brief description for each of the three steps below.

(1) Software defect information extraction. A bug report contains much information. However, not all of the information is useful. We are only interested in information that can be used to extract the software defect features for a bug triage. The goal of this step is filtering some irrelevant information, such as URL, target milestone, writeboard, keywords and so on. In this step, we complete the initial filter with stemming

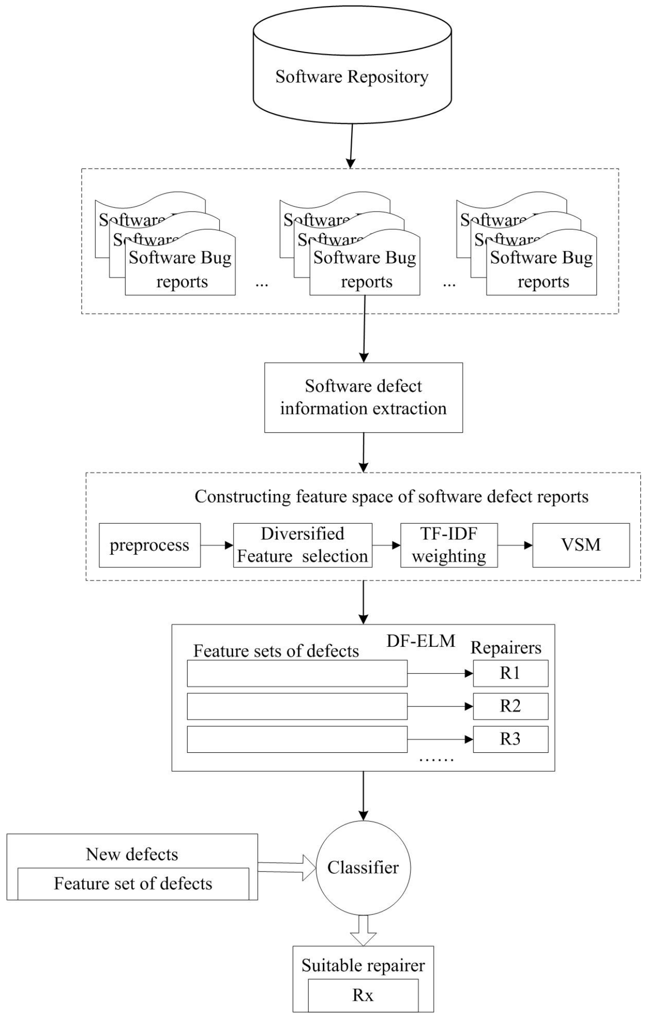


FIGURE 4. **Overview of defect triage.**

and removing of the stop words. More detailed information is given in subsection 4.1∼4.2.

(2) Diversified feature space construction. In this step, we use the defect information to construct a feature space of software defect reports. In this step, we mainly preprocessed the defect information, and finish the diversified feature selection of the representative features. The goal of this step is to select a representative group of nonredundant feature sets with relative maximal statistical information on the developer triage. More detailed information is given in section 4.3.

(3) Optimal ELM classification based on GA. Finally, based on the selected the diversity features, we train the ensemble ELM classifier. Fixers are as the labels used to train the ELM classifier. In this step, we develop an assembling ELM-based genetic algorithm to further promote the classifier's effectiveness. This method can output the mean value of a several weighted classifiers.

Instead of assigning equal weights to every classifier, for better-performing classifiers, we set their weight coefficients to be higher. In this paper, we use a genetic algorithm based method to find the optimal weight allocation coefficients for each classifier. When there are new software defects generated, we recommend the most suitable fixer according to the trained model. More details are given in section 4.4.

A. SOFTWARE DEFECT INFORMATION EXTRACTION

Most of the previous researchers extracted from the defect reports only information about the description and the corresponding contributors. However, we found that the summary and comments of the defect reports also provided substantial valuable software defect information. The summary briefly describes the basic types and locations of defects, and comments are the communication that the developer has made when repairing the software defects. Therefore, in addition to extracting the description and restorers' information of the defect reports, we also collect the summary and comments to improve the accuracy of the defect assignment.

In this paper, we use the defect tracking system Bugzilla to obtain the defect report data. In Bugzilla, a defect report records the details of a software defect. We extract the useful information, such as part of the summary, the description and the comments of the report as candidate attribute information for the software defect triage. The contributors were extracted as the labels of training samples. However, in Bugzilla, the contributors are not always assigned to the real fixers of software defects. For example, one defect was repaired by another developer, not the first developer it was assigned to, and it is also possible that the information regarding the assignment did not receive timely updates. First of all, we selected defect report data in the state of "solve". In order to obtain the real fixers of software defects, we adopt the following rules: (1). If a software defect is repaired by the developer that it is assigned to, then this developer is assumed to be the final real fixer of defect; (2). If a defect is not repaired by a developer it is assigned to, then the person who modifies the status of defect reports to "solve" eventually is assume to be the real fixer.

From the bug report, it can be seen that most of the bug report is about descriptive information and comments. However, the description and comments of software defect reports are in natural language descriptions, which contain much useless information. Even some noise affects the training effect of the classifier. The dimensional space may reach ten thousand dimensions or more when the text documents are represented by the vector space model. Thus, if we do not make perform data selection, it will be a complicated computation, which can not be tolerated in the actual classification. Therefore, it is important to preprocess the original data and select the representative features before training a classifier efficiently. There are two steps for preprocessing. The first step is stemming and the second step is removing the stop words.

Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form-generally a written word form. The stem of this word is not necessarily the same as the root of the word. The stem is usually sufficiently relevant to the root, even if this stem itself is not a valid root. When turning a text document into a vector space model VSM, the same word may has different forms in description, such as word form, i.e., like past tense and progressive tense, as described in natural language on the software defect reports. We must unify the word forms in the data preprocessing. Generally, there are some classic algorithms based on grammar rules, such as Porter stemmer and snowball stemmer. In this paper, we choose snowball stemmer for stemming.

The process of stemming extracts a word stem or root [4], mainly taking advantage of language rules to get the primary form of words. For example, in Table 1, the word "spending" turns into "spend", the word "created" turns into "create", and the word "keeps" turns into "keep". We translate the same or similar words into a consistent form after extracting stems and improved the validity of the selected feature. Further, the step also helps to reduce the dimension of the data to some extent.

TABLE 1. mapping relationship of the words and stems.

Original word	Stemmed word
spending	spend
created	creat
keeps	keep
deletion	delet
normally	normal
......	......

The system will filter out some words or phrases before or after handling natural language data, and these words or phrases are called stop words. Generally, stop words are the function words in the human language, and these function words are extremely common. Compared with other words, function words have no actual meanings, such as: "the", "is", "a", "at", "which", "that", "on", numbers, characters, punctuation, etc. Although these stop words cannot separately express the degree of correlation for a document, these stop words will take up considerable space. In the extracted software defect reports, most of the words are stop words which are not useful for the training classifier. The researcher named Ho, Tin Kam at Bell laboratory of American thought that "in a typical English article, stop words account for more than half, but the number of these stop words less than 150" [5]. The literature [6] lists the lists of stop words, and the number reaches 658, which includes many letter combinations but not words. Therefore, we first build a stop list in Table 2 and then filter data according to the stop list, which means that the data corresponding to the word are removed and that the data dimension and the data size are also reduced.

B. REPRESENTATIVE FEATURE SELECTION

There are much noise in bug reports. All-dimensional features will produce a high-dimensional vector if we map the

TABLE 2. parts of stop list.

a	b	c
able	back	come
about	become	can
above	been	contain
......	......	......

original data onto the vector space directly. Further, the noise also increases the complexity of time, the complexity of space and affects the accuracy of the classification [14], [15]. Therefore, efficient feature selection is an essential task for filtering those meaningless features, such as redundant features or irrelevant features, while retaining the important features. It is necessary for us to use a suitable feature selection method to reduce the feature space dimension and noise to improve the efficiency and accuracy of the classification. The common feature selection methods include mutual information (MI), information gain (IG), term strength (TS), chi-square (CHI). In the naive method, we choose IG as the feature selection method. The literature [36] claims that IG is one of the best feature selection methods. The formula of IG feature selection is as following:

$$IG(u) = P(u)\sum_{t} P(C_v/u)\log\frac{P(C_v/u)}{P(C_v)} + P(\overline{u})\sum_{t} P(C_v/\overline{u})\log\frac{P(C_v/\overline{u})}{P(C_v)} \quad (6)$$

where v is the total number of developer tags. $P(u)$ represents the probability of feature u. $P(C_v/u)$ represents the conditional probability of belonging to developer C_v class when the text contains feature u. $P(C_v)$ represents the probability of information text belonging to developer C_v in a text set. $P(\overline{u})$ represents the probability that feature u does not appear in the text. $P(C_v/\overline{u})$ represents the probability of belonging to the developer C_v class when the text does not contain feature u. In the experiments, we select a feature subset by the IG method to reduce the dimension of the vector space and filter the noise. In the next step, we weight the feature words of different frequencies to improve the triage accuracy of software defect reports.

If a word shows up in a paper with a high frequency, and rarely appears in other papers, this word has a very good ability to differentiate categories and is suitable for classification. In this case, we compute the weight-value for different important terms by using the TF-IDF method. The TF-IDF algorithm was proposed firstly in the literature by [7]. Later, Salton *et al.* [8] demonstrated the validity of TF-IDF repeatedly. TF-IDF is a statistical method used to evaluate the degree of the importance of a word for a document in a file set. For a given feature word t_i, the tf_{ij} of this word can be expressed as:

$$tf_{ij} = \frac{n_{i,j}}{\sum_k n_{k,j}} \quad (7)$$

In the above formula, $n_{i,j}$ is the number of times this feature word appears in the document d_j. The denominator represents the sum of the frequencies of all words that occur in the file. The idf_i of this word shown as:

$$idf_i = \log(1 + \frac{|D|}{1 + |j : t_i \in d_j|}) \tag{8}$$

$|D|$ is the total number of files in the file set. $|j : t_i \in d_j|$ represents the numbers of files including feature word t_i. If this feature word is not in the file set, then the denominator is zero, in this case, it is written as $1 + |\{j : t_i \in d_j\}|$. The weight $w_{i,j}$ of feature word t_i in file d_j can be expressed as:

$$w_{i,j} = \frac{tf_{i,j}}{max_j\{tf_{k,j}\}} \times idf_i \tag{9}$$

$max_j\{tf_{k,j}\}$ is the maximum number of feature words tf in file d_j. The weights of all the feature words with the above methods are calculated, and the process is normalized by establishing the vector space model (VSM).

After the processing of feature selection, we select the top-k features as the representative features. However, because these feature words are based on all the sample data, next, we must to establish the VSM of each sample data according to the feature words. During the construction of VSM, we utilize the following principles:

(1) Searching the data of each sample according to the feature words chosen. If the sample contains the feature words, then the VSM dimension value is 1; otherwise, the value is 0.

(2) Assigning values to the sample VSM according to the corresponding weight of each feature word calculated above. That means that if the dimension value is 1, the calculated value is the weight of the feature words. Otherwise, if the dimension value is 0, the value is unchanged.

V. OPTIMAL ELM ENSEMBLING BASED ON GA

When given some selected representative features, it is not difficult to train a two-class problem ELM classifier. However, many real-world data are multiclass and unbalanced [37]. Nevertheless, a multiclass problem can be solved by a primitive ELM with multiple learners. As we all know, an ensemble learner is much better than a single learner is, and ensemble strategies are more successful in many real-world domains [28]. Thus, in this paper, we proposed an ensemble of ELMs of multiple binary for lifting the accuracy.

In ELM, weights are set by random, however; weights also affect the accuracy of classification to some extent. In fact, the weights can be set higher for those classifiers performing well and lower for others. Therefore, finding the optimal weight coefficient can be transformed into optimization problem, which is difficult to be exactly settled especially when there are fewer feature dimensions. Using an intelligent optimization algorithm combined with ELM to extract the performance of high SLFNs has become a hot research topic [17]. In this paper, a genetic algorithm (GA) based method is proposed to explore the appropriate weight coefficients for each classifier.

A genetic algorithm is a random search optimization technique [11]. In GA, the search space is encoded in the form of *chromosomes*. All the chromosomes make up the population.

Each chromosome is associated with a fitness function representing the degree of superiority or inferiority of the chromosome.

Fitness function. Given a training instance r, the expected output of r is $d(r)$ and the actual output of the x-th individual ELM is $o_x(r)$. During the training, we expect that the output of the actual x should be consistent with the output of the running ELM. Moreover, let V_a be the validation set and $a = [a_1, a_2, \ldots, a_M]$ be a possible weight coefficient. According to the literature [12], the estimated generalization error of the ensemble ELM corresponding to a is:

$$E_a^W = \sum_{i=1}^{M}\sum_{j=1}^{M} a_i a_j C_{ij}^W = a^T C^W a, \tag{10}$$

where

$$C_{xj}^W = \frac{\sum_{r \in W} (f_x(r) - d(r))(f_j(r) - d(r))}{|W|} \tag{11}$$

where E_a^W denotes the goodness of a. The smaller E_a^W is, the better a is. Thus, we use the formula $f(a) = \frac{1}{E_a^W}$ as the fitness function.

Selection. During each successive generation, a certain selection method is required to judge the merits and demerits of a chromosome using a fitness function in order to select the best solution. In this paper, we adopt the roulette wheel as the selection strategy. Each chromosome associated with a selection probability. If f_t is the fitness of individual t in the population, the probability of an individual t being selected is

$$p_t = \frac{f_t}{\sum_{j=1}^{population} f_j} \tag{12}$$

where *population* is the total number of individuals in the population. In this case, chromosomes with lower fitness values are less likely to be selected, but there is still a small chance that they will.

Crossover. We adopt the normal single point crossover. The algorithm selects the crossover point between chromosomes randomly from a range from 0 to 1. The crossover probability is calculated with the same method used in [13]. Supposing that f_{maxfit} is the maximum fitness value on current population, the average fitness value of the population is $\overline{fit}$. Let fit' be the larger of the fitness values among the solutions. The crossover probability is δ_c.

$$\delta_c = \begin{cases} k_1 \times \dfrac{f_{maxfit} - fit'}{f_{maxfit} - \overline{f}}, & \text{if } fit' > \overline{fit}, \\ k_3, & \text{otherwise.} \end{cases} \tag{13}$$

where $k_1 = k_3 = 1.0$ [13]. The formula satisfies the following conditions: if $f_{maxfit} = \overline{fit}$, then $fit' = f_{maxfit}$ and $\delta_c = k_3$. The purpose of this is to achieve a balance between exploration and exploitation in a unique manner. The δ_c is upgraded while the better of the two chromosomes is crossed. The δ_c is low in order to decrease the probability of breaking off a good solution by crossover.

Mutation. Every chromosome has a process of mutation with a probability δ_m [13]. Chromosome mutation probability is also a manifestation of chromosome survival. That is, δ_m is given below:

$$\delta_m = \begin{cases} k_2 \times \dfrac{f_{maxfit} - f}{f_{maxfit} - \overline{f}}, & \text{if } fit > \overline{fit}, \\ k_4, & \text{otherwise}. \end{cases} \tag{14}$$

where the parameters k_2 and k_4 are set to 0.5.

Each position of the chromosome is mutated with probability δ_m. The value is replaced using a random variable drawn from a Laplacian distribution, $p(\epsilon) \propto e^{-\frac{|\epsilon-\alpha|}{\beta}}$, where the parameter β is the perturbation magnitude and α is the value of the position to be disturbed. β is equal to 0.1. The newly generated value is in place of the old value at the position. Once we generate a random variable using Laplacian distribution, there is a probability of generating a value that is closer to the old value.

The GA process of computing fitness value, selection, crossover and mutation is carried out in iteration. The optimal chromosome supplies the solution with the suitable weighted coefficient for the ensemble ELM classifier. The sum of the $weight_i$ should be saved during the evolving. Therefore, we normalized the evolved $weight$. The process of normalization is: $weight_i = weight_i / \sum_{i=1}^{M} weight_i$.

VI. EXPERIMENTAL ANALYSIS

DATASETS

The experiments are conducted on both real datasets. IOT software can be submitted by many bug report tracking systems. In this paper, we collected four bug repositories including Bugzilla,[1] Eclipse,[2] GCC[3] and Netbeans[4] to to examine the performance of the ELM classifier effectively. Table 3 summarizes the characteristics of the four datasets: the number of samples (# bug reports), the average bug reports for each developer (# B.D), and the average comments of each developer i(# C.D). For every experiment, we select 66% of the bug reports as training sets, and 34% as testing sets.

[1]https://bugzilla.mozilla.org/
[2]https://bugs.eclipse.org/bugs/
[3]https://gcc.gnu.org/bugzilla/buglist.cgi?
[4]https://netbeans.org/bugzilla/

TABLE 3. The real dataset information.

Data Set	# of bug reports	Average # of B.D	Average # of C.D
Bugzilla	5200	62	5.65
Eclipse	4100	43	5.26
Netbeans	3824	22	5.2
GCC	2102	32	4.34

In this set of experiments, we conduct the effectiveness analysis by comparing the classification accuracies over a series of combinations of different classifiers and feature selection methods. For each table in this subsection, the rows correspond to different classifiers, which are ELM, SVM, NaiveBayes, C4.5 and KNN the columns correspond to different feature selection methods, which are DF (*abbr.* of diverfied features), IG (*abbr.* of information gain), TR (*abbr.* of twoing rule), SM (*abbr.* of sum minority), MM (*abbr.* of max minority), GI (*abbr.* of gini index). Each entry corresponds to a different combination of the corresponding classifier and feature selection method, the accuracy of which is recorded in this entry. *Note*: Diversified features here refers in particular to the discovered features by our method in Section 4. The other five feature selection methods are available in book [9]. The methods are widely used in machine learning and are used as comparative methods in many feature selection studies on various domains.

Tables 4 ∼ 7 give the comparisons on the four real datasets. As seen from the four tables, the DF-based approach always provides the highest accuracy on all classifiers. This is mainly because the features selected by DF can be considered as providing the smallest but most representative coverage of all the

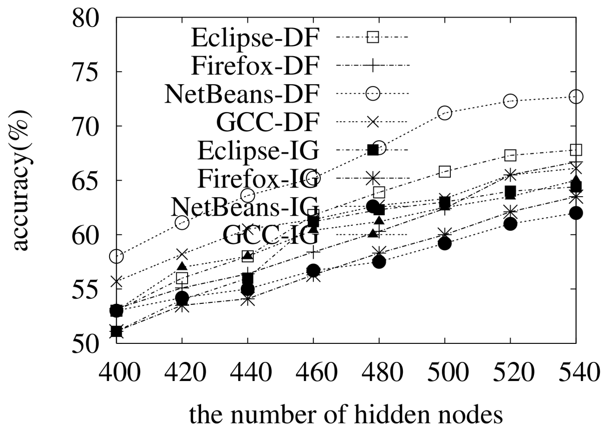


FIGURE 5. Accuracy vs hidden nodes.

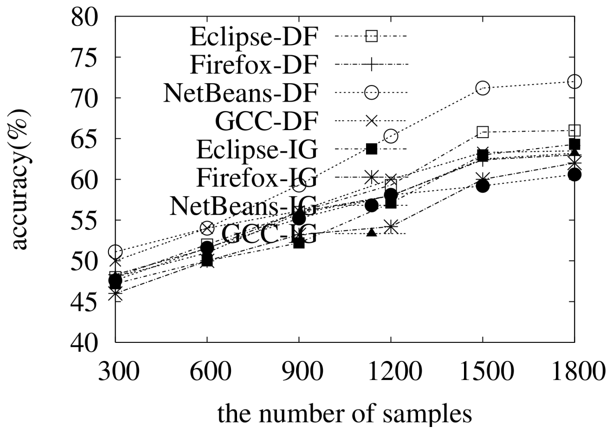


FIGURE 6. Accuracy vs samples.

TABLE 4. Accuracy on Eclipse vs #feature selection.

	DF	IG	TR	SM	MM	GI
ELM	65.8%	61.7%	59.3%	57.6%	58.8%	51.2%
SVM	56.07%	54.4%	55.8%	54.6%	53.1%	55.5%
Naive Bayes	46.6%	44.5%	40.8%	48.2%	49.4%	50.8%
C4.5	57.9%	53.8%	57.2%	53.7%	53.2%	53.5%
KNN	49.5%	48.3%	48.5%	49.2%	50.5%	48.8%

TABLE 5. Accuracy on Firefox vs #feature selection.

	DF	IG	TR	SM	MM	GI
ELM	62.4%	56.3%	53.7%	52.9%	53.7%	49.4%
SVM	56.9%	56.2%	58.4%	55.3%	54.6%	54.8%
Naive Bayes	41.2%	40.6%	37.7%	39.8%	40.1%	44.6%
C4.5	59.1%	56.7%	55.2%	53.8%	52.4%	51.6%
KNN	51.9%	50.1%	51.3%	51.2%	51.7%	51.6%

TABLE 6. Accuracy on Netbeans vs #feature selection.

	DF	IG	TR	SM	MM	GI
ELM	71.2%	68.7%	62.4%	59.6%	61%	60.2%
SVM	60.8%	53.6%	54.3%	53.2%	52.9%	53.7%
Naive Bayes	41.7%	38.9%	34.8%	31.4%	32.1%	36.8%
C4.5	60.2%	56.8%	57.5%	55.73%	56.6%	54.4%
KNN	53.8%	53.3%	53.7%	54.8%	54.3%	53.6%

TABLE 7. Accuracy on GCC vs #feature selection.

	DF	IG	TR	SM	MM	GI
ELM	63.3%	61.6%	60.2%	61.1%	63.3%	61.5%
SVM	55.6%	53.7%	52.5%	54.6%	55.2%	55.2%
Naive Bayes	39.4%	38.8%	37.8%	35.6%	39.5%	36.2%
C4.5	55.6%	52.5%	52.6%	54.6%	51.2%	51.6%
KNN	46.2%	45.4%	43.6%	44.8%	45.3%	45.8%

original features. They provide less redundant information than the other methods, while taking into account the feature interactions.

Figures 5 ∼ 7 present the effectiveness comparison of four different data sets, including Bugzilla, Eclipse, GCC and Netbeans. The effectiveness is evaluated by comparing how the accuracy varies with the number of samples, hidden nodes, and features. Figure 5 shows the accuracy comparison with different hidden nodes changing while the number of features is fixed at 500 and the number of samples is fixed at 1500. Figure 6 shows the accuracy comparison when changing the number of different samples while the number of hidden nodes is fixed at 500 and the number of samples is fixed at 1500. Figure 7 shows the accuracy comparison when changing the number of features while the number of samples is fixed at 1500 and the number of hidden nodes is fixed at 500.

As seen from the results, no matter how the number of samples, features or hidden nodes varies, DF-ELM is constantly better than IG-ELM. This outcome is because DF-ELM is based on a diversified feature selection. It explicitly performs a removing of the irrelevant and the redundant features. However, IG-ELM, ELM with feature selection by IG, runs

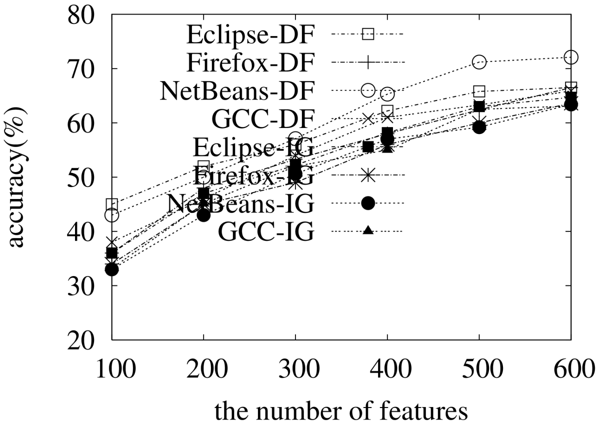


FIGURE 7. Accuracy vs features.

on the selected features, which makes it have an increasing chance of being affected by overfitting. The observation that DF-ELM is more accurate than IG-ELM is consistent with the fact that an ensemble is usually more accurate than a single learner is [28].

The time-consuming situation when a classifier model is constructed by using a training set is shown in

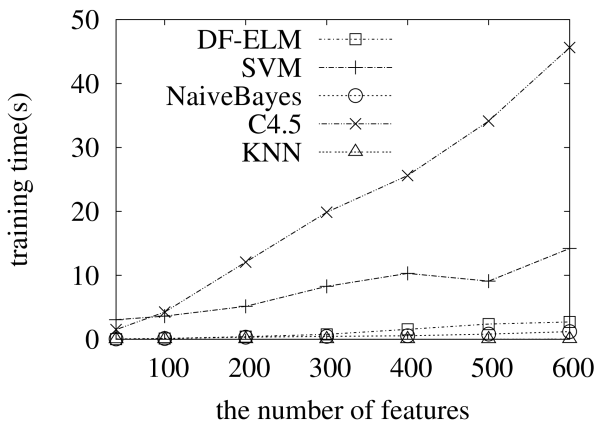


FIGURE 8. Training times(s) on Eclipse.

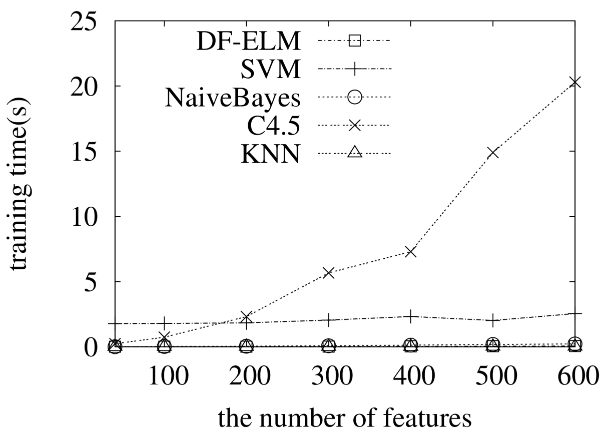


FIGURE 11. Training times(s) on GCC.

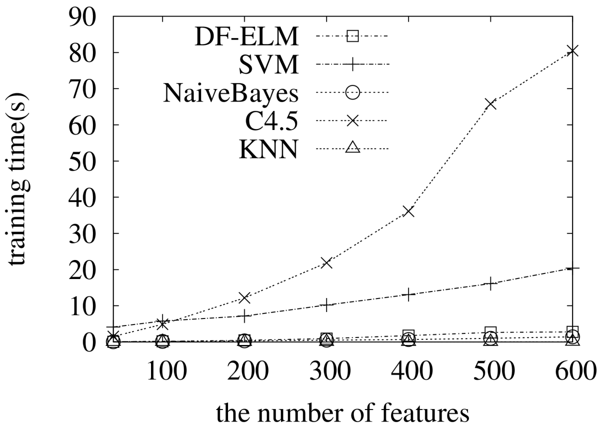


FIGURE 9. Training times(s) on Firefox.

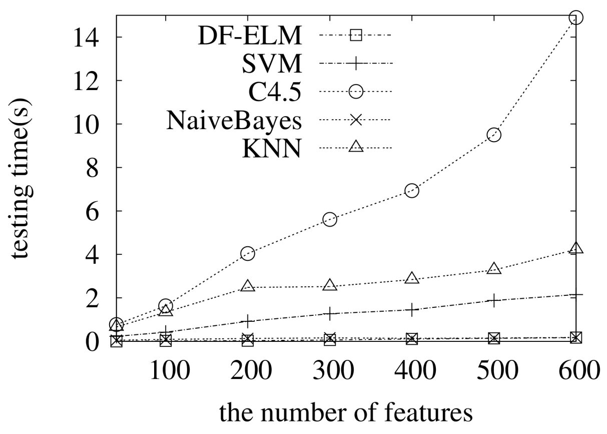


FIGURE 12. Testing times(s) on Eclipse.

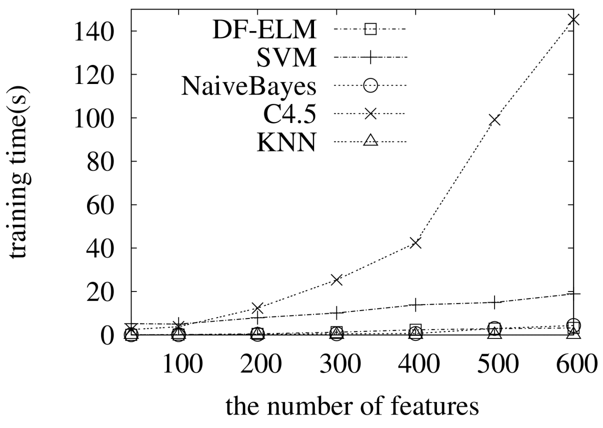


FIGURE 10. Training times(s) on Netbeans.

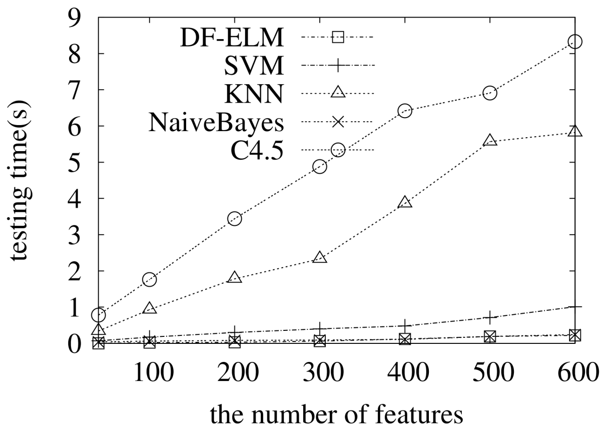


FIGURE 13. Testing times(s) on Firefox.

Figure 8 ∼ Figure 15. Figure 8 ∼ Figure 11 shows the training times on different datasets and Figure 12 ∼ Figure 15 shows the testing times on different datasets. Taking the variety of features as an example, the training time is increasing as the number of data features increases. We concluded that DF-ELM is comparable to SVM, Naive Bayes, C4.5 and KNN on accuracy while using much less training time. Thus, DF-ELM has a better tradeoff between effectiveness and efficiency than that of SVM, Naive Bayes, C4.5 and KNN.

Through the previous analysis, we have already known that, in most cases, ELM-based classifiers are the least

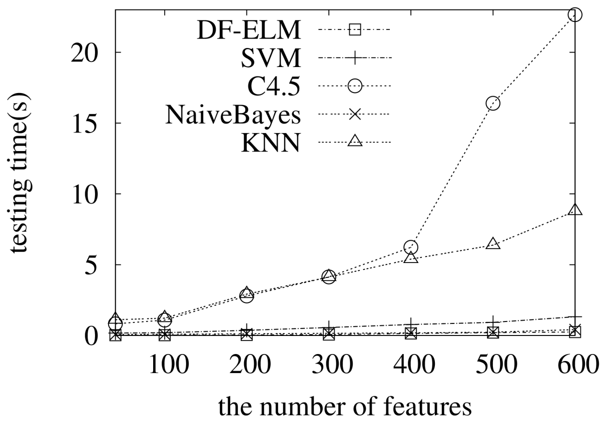


FIGURE 14. **Testing times(s) on Netbeans.**

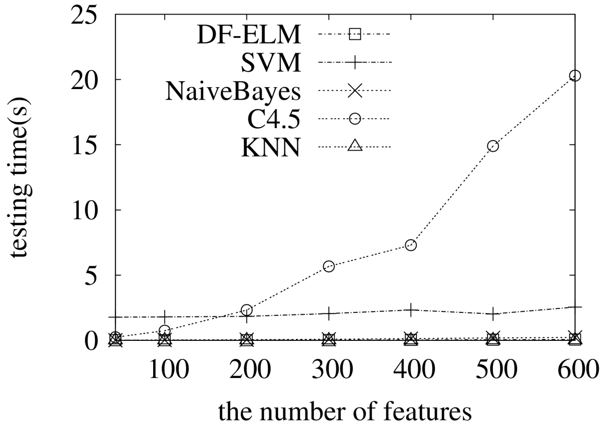


FIGURE 15. **Testing times(s) on GCC.**

time-consuming in terms of training. It can be seen that the DF-ELM-based classifier has the shortest testing time and that the size of the data does not have much effect on the testing time of DF-ELM classifier. However, C4.5-based classifier training takes the longest time in most cases. DF-ELM-based classifiers and Naive Bayes-based classifiers consume almost the same testing time in most cases.

VII. CONCLUSION

With the rapid development of the Internet of Things industry in the world, trouble-free software has played a greater role. The rapid and efficient elimination of bugs in the software is vital to the growth of Internet of Things enterprises. In this paper, we propose a software defect triage framework based on ELM. Through experiments, we conclude that the effects of the classification and the time consumption of the defect triage classifier based on ELM are much better than those of classifiers based on other algorithms. In future work, we will improve the ELM-based classifier. In this paper, when we assign a defect, each defect is assigned only to a specific restorer, which reduces the accuracy of the defect assignment because the software defects that can be repaired do not significantly differ between restorers in real of software development; in other words, a software defect can be repaired by multiple restorers. Thus, in the future, we will fully consider the capability of multiply restorers to repair defects and to recommend a number of the most appropriate restorers for each software defect, thereby improving the accuracy of defect assignment, especially for the growing of IOT enterprises.

REFERENCES

[1] N. S. M. Yusop, J. Grundy, and R. Vasa, "Reporting usability defects: A systematic literature review," *IEEE Trans. Softw. Eng.*, vol. 43, no. 9, pp. 848–867, Sep. 2017.

[2] N. S. M. Yusop, "Understanding usability defect reporting in software defect repositories," in *Proc. ASWEC*, vol. 2, 2015, pp. 134–137.

[3] W. Yu *et al.*, "A survey on the edge computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[4] G. Nicolai and G. Kondrak, "Leveraging inflection tables for stemming and lemmatization," in *Proc. ACL*, vol. 1, 2016, pp. 1138–1147.

[5] T. K. Ho, "Stop word location and identification for adaptive text recognition," *Int. J. Document Anal. Recognit.*, vol. 3, no. 1, pp. 16–26, 2000.

[6] (Jun. 14, 2007). *Stop Word List-Words Filtered out by Search Engine Spiders*. [Online]. Available: http://www.twitterbuttonfactory.com/www/seo-innovation.com/

[7] G. Salton and C. T. Yu, "On the construction of effective vocabularies for information retrieval," in *Proc. SIGIR*, 1973, pp. 48–60.

[8] G. Salton, E. A. Fox, and H. Wu, "Extended Boolean information retrieval," *Commun. ACM*, vol. 26, no. 11, pp. 1022–1036, 1983.

[9] I. H. Witten, F. Eibe, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Mateo, CA, USA: Morgan Kaufmann, 2011, pp. 1–629.

[10] Z. Wang, Y. Zhao, G. Wang, Y. Li, and X. Wang, "On extending extreme learning machine to non-redundant synergy pattern based graph classification," *Neurocomputing*, vol. 149, pp. 330–339, Feb. 2015.

[11] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA, USA: Addison-Wesley, 1989, pp. 1–140.

[12] Z.-H. Zhou, J.-X. Wu, Y. Jiang, and S.-F. Chen, "Genetic algorithm based selective neural network ensemble," in *Proc. 17th Int. Joint Conf. Artif. Intell. (IJCAI)*, Seattle, DC, USA, 2001, pp. 797–802.

[13] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 4, pp. 656–667, Apr. 1994.

[14] M. Radovic, M. Ghalwash, N. Filipovic, and Z. Obradovic, "Minimum redundancy maximum relevance feature selection approach for temporal gene expression data," *BMC Bioinf.*, vol. 18, no. 1, pp. 9:1–9:14, 2017.

[15] D. Zuckerman, "On unapproximable versions of *NP*-complete problems," *SIAM J. Comput.*, vol. 25, no. 6, pp. 1293–1304, 1996.

[16] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, nos. 1–3, pp. 489–501, 2006.

[17] G. Huang, G.-B. Huang, S. Song, and K. You, "Trends in extreme learning machines: A review," *Neural Netw.*, vol. 61, pp. 32–48, Jan. 2015.

[18] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proc. Int. Conf. Softw. Eng.*, Shanghai, China, 2006, pp. 361–370.

[19] G.-B. Huang, Y.-Q. Chen, and H. A. Babri, "Classification ability of single hidden layer feedforward neural networks," *IEEE Trans. Neural Netw.*, vol. 11, no. 3, pp. 799–801, May 2000.

[20] R. K. Roul, A. Nanda, V. Patel, and S. K. Sahay, "Extreme learning machines in the field of text classification," in *Proc. SNPD*, Jun. 2015, pp. 217–223.

[21] D. D. Wang, R. Wang, and H. Yan, "Fast prediction of protein–protein interaction sites based on extreme learning machines," *Neurocomputing*, vol. 128, pp. 258–266, Mar. 2014.

[22] G. Wang, Y. Zhao, and D. Wang, "A protein secondary structure prediction framework based on the extreme learning machine," *Neurocomputing*, vol. 72, nos. 1–3, pp. 262–268, 2008.

[23] T. Helmy and Z. Rasheed, "Multi-category bioinformatics dataset classification using extreme learning machine," in *Proc. IEEE Congr. Evol. Comput.*, May 2009, pp. 3234–3240.

[24] L. Scandolo, P. Bauszat, and E. Eisemann, ''Merged multiresolution hierarchies for shadow map compression,'' *Comput. Graph. Forum*, vol. 35, no. 7, pp. 383–390, 2016.
[25] K. Ning, M. Liu, and M. Dong, ''A new robust ELM method based on a Bayesian framework with heavy-tailed distribution and weighted likelihood function,'' *Neurocomputing*, vol. 149, pp. 891–903, Feb. 2015.
[26] G.-B. Huang, X. Ding, and H. Zhou, ''Optimization method based extreme learning machine for classification,'' *Neurocomputing*, vol. 74, pp. 155–163, Dec. 2010.
[27] G.-B. Huang, D. H. Wang, and Y. Lan, ''Extreme learning machines: A survey,'' *Int. J. Mach. Learn. Cybern.*, vol. 2, no. 2, pp. 107–122, Jun. 2011.
[28] N. Settouti, M. A. Chikh, and V. Barra, ''A new feature selection approach based on ensemble methods in semi-supervised classification,'' *Pattern Anal. Appl.*, vol. 20, no. 3, pp. 673–686, 2017.
[29] T. Zhang, H. Jiang, X. Luo, and A. T. S. Chan, ''A literature review of research in bug resolution: Tasks, challenges and future directions,'' *Comput. J.*, vol. 59, no. 5, pp. 741–773, 2016.
[30] T. Zhang, G. Yang, B. Lee, and A. T. S. Chan, ''Guiding bug triage through developer analysis in bug reports,'' *Int. J. Softw. Eng. Knowl. Eng.*, vol. 26, no. 3, pp. 405–432, 2016.
[31] E. A. Felix and S. P. Lee, ''Integrated approach to software defect prediction,'' *IEEE Access*, vol. 5, pp. 21524–21547, 2017.
[32] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo. (2017). ''Automatic bug triage using semi-supervised text classification.'' [Online]. Available: https://arxiv.org/abs/1704.04769
[33] J. Xuan *et al.*, ''Towards effective bug triage with software data reduction techniques,'' *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 264–280, Jan. 2015.
[34] S. N. Ahsan, J. Ferzund, and F. Wotawa, ''Automatic software bug triage system (BTS) based on latent semantic indexing and support vector machine,'' in *Proc. 14th Int. Conf. Softw. Eng. Adv.*, Sep. 2009, pp. 216–221.
[35] D. Cubranic and G. C. Murphy, ''Automatic bug triage using text categorization,'' in *Proc. 16th Int. Conf. Softw. Eng. Knowl. Eng.*, Jun. 2004, pp. 92–97.
[36] H. T. Le, T. Urruty, S. Gbèhounou, F. Lecellier, J. Martinet, and C. Fernandez-Maloigne, ''Improving retrieval framework using information gain models,'' *Signal, Image Video Process.*, vol. 11, no. 2, pp. 309–316, 2017.
[37] S. Huda *et al.*, ''An ensemble oversampling model for class imbalance problem in software defect prediction,'' *IEEE Access*, vol. 6, pp. 24184–24195, 2018.

YING YIN (M'11) received the B.E., M.E., and Ph.D. degrees in computer science from Northeastern University, China, in 2002, 2005, and 2008, respectively. He is currently an Associate Professor with the School of Information Science and Engineering, Northeastern University. His major research interests include data mining and machine learning. He is a member of ACM and CCF.

XIANGJUN DONG received the M.E. degree in computer applications from Shandong Industrial University in 1999 and the Ph.D. degree in computer applications from the Beijing Institute of Technology in 2005. He is currently a Professor with the School of Information, Qilu University of Technology, Jinan, China. He has published research papers in national and international journals and in conference proceedings. His research interests include association rules, sequential pattern mining, and negative sequential pattern mining.

TIANTIAN XU received the B.E. and M.E. degrees in computer applications from the Qilu University of Technology in 2012 and 2015, respectively. She has published research papers in international journals. Her research interests include pattern recognition, association rules, and sequential pattern mining.

• • •