Artificial Life and Robotics (2021) 26:318–328
https://doi.org/10.1007/s10015-021-00681-3

ORIGINAL ARTICLE

A method of non-bug report identification from bug report repository

Jantima Polpinij[1]

Received: 15 April 2020 / Accepted: 12 February 2021 / Published online: 20 March 2021

Abstract
One of the most common issues addressed by bug report studies is misclassification when identifying and then filtering non-bug reports from the bug report repository. Having to filter out unrelated reports wastes time in identifying actual bug reports, and this escalates costs as extra maintenance and effort are required to triage and fix bugs. Therefore, this issue has been seriously studied and is addressed here. To tackle this problem, this study proposes a method of automatically identifying non-bug reports in the bug report repository using classification techniques. Three points are considered here. First, the bug report features used are unigram and CamelCase, where CamelCase words are used for feature expansion. Second, five term weighting schemes are compared to determine an appropriate term weighting scheme for this task. Lastly, the support vector machine (SVM) family i.e. binary-class SVM, one class SVM based on Schölkopf methodology and support vector data description (SVDD) are used as the main mechanisms for modeling non-bug report identifiers. After testing by recall, precision, and F1, the results demonstrate the efficiency of identifying non-bug reports in the bug report repository. Our results may be acceptable after comparing to the previous well-known studies, and the performance of non-bug report identifiers with *tf-igm* and modified *tf-icf* weighting schemes for both Scölkopf methodology and SVDD methods yielded the best value when compared to others.

Keywords Bug reports · Non-bug report identifier · Text classification · Support vector machine (SVM) · Schölkopf methodology · Support vector data description (SVDD)

1 Introduction

Many very large and complex open sources or software application projects have been proposed [1–8], but no software is completely safe from defects, also known as "*bugs*" [5]. In general, the software testing process locates bugs or defects in a program. However, it is impossible to locate all the bugs in a piece of software. End users can be employed as testers to locate and identify bugs in software. Information relating to software problems reported by software testers and end users is termed as a "*bug report*". Bug reports contain key information for maintaining and enhancing software efficiency and quality. Thus, it is not wondering if numerous software projects utilizing bug reports as guideline for the maintenance task. Consequently, utilizing bug reports may have helped to reduce maintenance cost. It is well-known that this cost is the highest in software development life cycle [2].

Bug tracking systems (BTS) have been developed as a bug tracking tool that is used for gathering a large number of bug reports, comments, and additional requirements from more users [1–10]. Now, many BTSs like Bugzilla, Mantis, Redmine, FogBugz, Airbrake, Backlog, Trac, YouTrack, or Jira are widely used [1–8]. When a new bug report is sent to the bug report repository via the BTS, software experts that are called "*bug triager*" analyze, classify, and prioritize the report before assigning suitable developers to fix a bug mentioned in the report [2, 3, 5, 7, 8]. Unfortunately, these tasks are time-consuming when manually working [1–11]. This leads the concept to handle this problem with automatic analysis way. As a result, many studies related to bug reports have been proposed. These studies can be classified into three main areas: bug report optimization, bug report triage, and bug fixing [6]. Bug report optimization concentrates to

This work was presented in part at the 25th International Symposium on Artificial Life and Robotics (Beppu, Oita, January 22–24, 2020).

✉ Jantima Polpinij
jantima.p@msu.ac.th

[1] Faculty of Informatics, Mahasarakham University, Mahasarakham 44150, Thailand

enhance the quality of the report, filtering irrelevant reports, and reduce incorrect information. Bug report triage aims to reduce duplicate bug report, prioritize bug reports, and assign suitable software developer for fixing bugs. At last, bug fixing is related to debug and recover links between the bug reports and corresponding changes.

One of the most common issues addressed by bug report studies is to first identify and then filter non-bug reports from the bug report repository. Bug triggers waste time by having to filter out unrelated reports and identify actual bug reports, and this escalates costs as extra maintenance time and effort in triaging and fixing bugs [10, 12–14]. Therefore, this issue has been seriously studied. Also, it is a challenge for this study.

To tackle this problem, this study aims to propose a method of automatically identifying non-bug reports in the bug report repository using classification techniques, where the main mechanism of this classification is one-class support vector machines (OC-SVM). The OC-SVM is applied with several term weighting schemes.

The rest of this paper is organized as follows. Section 2 is the literature review, while Sect. 3 describes the datasets used in this study and Sect. 4 presents research methodology. The experimental results are given in Sect. 5. Finally, a conclusion is in Sect. 6.

2 Literature review

Bug reports describe problems, especially in open-source software. Herzig et al. [12] suggested that an issue can be classified as a '*bug*' or a '*defect*' if it requires corrective code maintenance. However, some bug reports that are classified as non-bug often mention for perfective and adaptive maintenance, commentating, complaining, refactoring, discussions, and so on. Therefore, quality of bug reports is necessary because the development team used this information from bug reports to find and track the issues in a particular software. Simply speaking, information in '*actual*' bug reports can determine the software maintenance efficiency and software fixing time reduction. Previous studies reported that researchers spent 90 days manually classifying more than 7,000 bug reports as a time-consuming task [10–12]. After manual classification, 39% of the bug reports initially marked as 'defective' never had a bug [12]. This issue was termed as a "*misclassification*" between bug and non-bug reports [10, 12, 13]. Consequently, many bug report studies have proposed the adoption of automated analysis methods.

The first study of automated bug analysis was conducted by Antoniol et al. [10]. They applied machine learning algorithms namely Decision Trees (DT), Logistic Regression (LR), and Naïve Bayes (NB) to develop text classifiers that automatically distinguished bug and non-bug reports. Their results indicated that the accuracy of classifying bug reports from three open sources (i.e. Mozilla, Eclipse, and JBoss) was between 0.77 and 0.82.

In 2013, Herzig et al. [12] manually analyzed more than 7000 bug reports downloaded from Bugzilla and Jira. They found that one-third of the bug reports that were analyzed as actual-bug reports were non-bugs. As a result, they generated a standard dataset, called Herzig's dataset that has subsequently been used in many studies [12–16].

Pingclasai et al. [13] proposed a method based on topic modeling using Latent Dirichlet Allocation (LDA) to find the most efficient models using three open sources as HttpClient, Jackrabbit, and Lucene containing 745, 2402 and 2443 bug reports, respectively (derived from Herzig's analysis). This study compared three classification algorithms as DT, LR, and NB. Furthermore, Pingclasai et al. also compared the classifying performance between a topic-based model and a word-based model. Results gave F1 scores between 0.65 and 0.82, with NB classifiers determined as the highest performance model.

Limsetho et al. [14] proposed a method to automatically cluster bug reports, and label these clusters based on their textual information without the need for training data. Two unsupervised learning algorithms namely Expectation Maximization (EM) and X-Means were applied. Similar bug reports were grouped and automatically given labels with meaningful and representative names. This study used three bug report datasets namely Lucene, Jackrabbit, and HTTPClient from [12]. Experimental results showed that this framework achieved performance comparable to supervised learning algorithms (i.e. J48 and LR). Limsetho et al. concluded that their framework was suitable for use as an automated categorization system that could be applied without prior knowledge.

In 2017, Terdchanakul et al. [15] proposed a solution for the bug report misclassification problem. They used N-gram IDF as an extension of IDF to manage terms or phrases of different lengths that were used as features of the documents using the data set from [12] and applied LR and Random Forest (RF) algorithms to model the classification. The experiment compared the use of N-gram IDF to topic-based models. Their proposed method returned F1 scores between 0.79 and 0.81, with 10-fold cross validation in LR and RF techniques, respectively. Furthermore, Qin and Sun [16] studied the same problem. They proposed a bug classification method based on a typical recurrent neural network (RNN). They performed the existing topic-based method and N-gram IDF-based method on four datasets, including Herzig's data set [12]. Results showed F1 score at 0.746 and superior to N-gram values. They suggested that their research might assist developers and researchers to classify bug reports and help to identify misclassified bug reports.

3 Datasets

This study used two datasets. The first was a standard dataset, called the Herzig's dataset [12]. We utilized the "*bug summary*" to analyze and classify bug reports into actual-bug and non-bug classes because this part contained less noise [2, 17, 18]. Therefore, many studies related to bug reports consider only the summary part. Here, this work also uses the summary part. An example of bug reports is presented as Fig. 1.

The other dataset was downloaded from Bugzilla (https://www.bugzilla.mozilla.org/). Bug reports relating to Mozilla Firefox were downloaded on November 1, 2019. This dataset consisted of 10,000 bug reports. Then, 5000 bug reports labeled with "*verified*" and "*closed*" were selected because they were already confirmed by a software development team as actual-bug reports, while the other 5000 bug reports were labeled with "*invalid*" status. Finally, this dataset can be summarized and shown in Table 1.

Table 1 Summary of the datasets

Dataset	Software	Bug	Non-bug
Herzig's dataset	Jackrabbit	937	1464
Real-world dataset	Firefox	5000	5000

4 The methodology

The proposed research methodology consists of three main processing steps. They are bug report pre-processing, bug report representation and term weighting and non-bug report identifier modeling. Each step is presented in more detail as follows.

4.1 Bug report pre-processing

First, the training set separates text into words using word delimiters (e.g. white space), and then the stop-words are removed. In this study, the bug report features (or words) used are a combination of unigram and CamelCase. In [10,

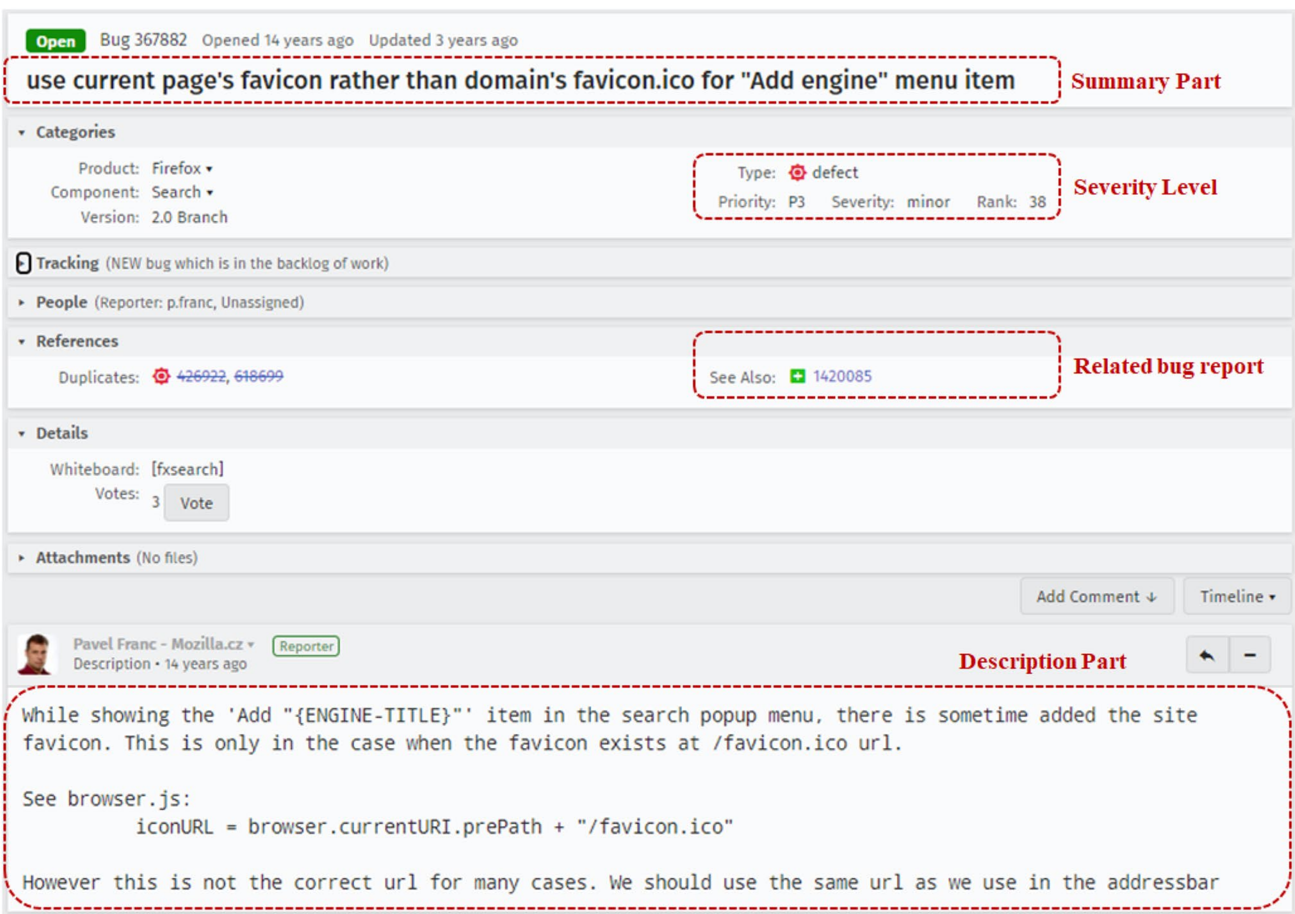

Fig. 1 An example of bug report

19–21], they demonstrated that the use of unigram and CamelCase return satisfactory results in the study of bug reports. This is because unigram words can generally be found in any bug report, while the CamelCase words indicate the specificity of the software. Using CamelCase, this is to expand keywords and helps to increase the search efficiency [22].

Later, the process of stop-word removal takes place. After removing the stop-words, punctuation is removed, and some word forms are changed into proper ones as shown in Table 2.

It is noted that bug report featured are also selected by using *information gain* (*IG*) with threshold as 0.2. Simply speaking, if a term weight score is less than 0.2, that term should be ignored. After ranking the IG scores, the keywords in the top 20, 50, 100, and 150 are selected as the bug report features.

4.2 Bug report representation and term weighting

After pre-processing, the bug reports are expressed as a vector representation, called a *bag-of-words* (BoW). A BoW is used to describe the occurrence of words within a textual document. After transforming the text into a BoW, the next process is to calculate various measures to characterize the text, called term weighting. Here, five term weighting methods are compared to obtain the most suitable.

These term weighting schemes are *tf* (term frequency), *tf-idf* (term frequency-inverse document frequency), *tf-igm* (term frequency-inverse gravity moment), *tf-icf* (term-frequency inverse class frequency) and modified *tf-icf*.

4.2.1 Term-frequency (tf)

tf shows how frequently a term-word occurs in a bug report. In general, it is often useful to skew normalization using a logarithmic scale. The formula for *tf* is represented as:

$$tf = log(1 + f_{t,d}) \tag{1}$$

The *tf* weighting scheme is often used in the context of bug reports because it has been mentioned that it returns satisfactory analysis results [10].

Table 2 Examples of word normalization

Original form in bug report	Normalized Form	Original form in bug report	Normalized form
Didn't	Did not	Can't	Can not
Don't	Do not	's	Is

4.2.2 Term frequency-inverse document frequency (tf-idf)

tf-idf consists of local weigh (*tf*) and global weight (*idf*) [23]. The formula of *tf-idf* is represented as:

$$tf\text{-}idf = log(1 + f_{t,d}) \times log\left(1 + \frac{N}{df_t}\right) \tag{2}$$

where N is the whole number of bug reports appearing in the dataset and df_t is the number of bug reports containing term t.

4.2.3 Term frequency- inverse gravity moment (tf-igm)

The third term weighting scheme is *tf-igm* introduced by Chen et al. [24] as a supervised term weighting scheme. It modifies and improves *tf-idf*. The *tf-igm* can calculate the distinguishing class of a term precisely. Its formula is:

$$tf\text{-}igm_{t,d} = f_{t,d} \times (1 + \lambda \times igm(t_k)) \tag{3}$$

where $t_{t,d}$ is the frequency of term t occurring in document d, and λ (Lambda) is defined as an adjustable coefficient factor used to achieve relative balance between $t_{t,d}$ and *igm* factors in the weight of term t. The default value of λ is 7.0 but it can be set as a value between 5.0 and 9.0 [24]. For *igm* factor is used to calculate the inter-class distribution concentration of a term. The *igm* formula is:

$$igm(t_k) = \frac{f_{k1}}{\sum_{r=1}^{m} f_{kr} \times r} \tag{4}$$

where f_{k1} represents the frequency of term t_k in the class in which it occurs most often, while $f_{k,r}$ $(r = 1, 2, ..., m)$ are the frequencies of t_k that occur in different classes in descending order, with r defined as the rank. Simply speaking, the frequency f_{kr} refers to the class-specific document frequency (*df*). It is the number of documents in the r-th class that contain the term t_k and it is denoted as $df_{k,r}$.

4.2.4 Term frequency- inverse class frequency (tf-icf)

tf-icf is a modification of *tf-idf* proposed by Lertnattee and Leuviphan [25]. They replaced the *idf* factor by *icf*, where *icf* might represent importance of information among classes. The *tf-icf* can be formulated as:

$$tf - icf = f_{t,d} \times log_2\left(\frac{|C|}{cf_t}\right) \tag{5}$$

where *tf* is the frequency of term t found in a document d, while $|C|$ is the whole number of classes and cf_t is the number of classes that include the term t.

4.2.5 Modified tf-icf

A term may occur in many classes, but the importance of that term may be different in each class. Therefore, it was modified here, where the modified *tf-icf* is able to measure the class distinguishing power of a term. The formula is defined as:

$$\text{modified } tf\text{-icf } = f_{t,d} \times log_2\left(\frac{|N_c|}{df_{t,c}}\right) \tag{6}$$

where N_c is the whole number of bug reports in class c, and $df_{t,c}$ is the number of bug reports in class c containing the term t. This may help to measure the importance of each word in the distinguishing class.

4.3 Non-bug report identifiers modeling

To model a non-bug report identifier, we applied the support vector machines (SVM) family. This is because SVM works relatively well and uses memory efficiently. This algorithm maximizes the margin of the decision boundary using quadratic optimization techniques to find the optimal hyperplane. This algorithm is more effective in high-dimensional feature spaces [26]. However, The SVM algorithm is not suitable for large datasets and does not work well if the dataset has excessive noise. SVM algorithm was chosen because its limitations are relevant to the characteristics of our datasets used in this study. These bug report datasets are quite small, and each bug report contains less text because only the '*summary part*' of the bug report was used. Although this part contains less text, it has been confirmed by many previous studies that it may contain less noise [2, 17, 18]. Therefore, we expected the SVM family to work well in this study.

4.3.1 Traditional binary-class SVM

The fundamental of SVM is to create a function that takes the value +1 in a "*relevant*" region capturing most of the data points (called *support vectors*) that are closer to the hyperplane, and -1 elsewhere [26]. Learning can be regarded as finding the maximum margin separating the hyperplane between two classes of points. Suppose that a pair (w, b) defines a hyperplane which has the following formula [26].

$$f(x) = wx + b \tag{7}$$

Then, a normalization is chosen such that $w > x^+ + b = +1$ and $w > x^- + b = -1$ for the positive and negative support vectors, respectively. The margin can be given by:

$$\frac{w}{\|w\|}(x^+ - x^-) = \frac{w^T(x^+ - x^-)}{\|w\|} = \frac{2}{\|w\|} \tag{8}$$

Learning the SVM can be defined as a following optimization:

$$\max_{w} \frac{2}{\|w\|} \tag{9}$$

Subject to:

$$\begin{aligned} w^T x_i + b \geq +1 \text{ if } y_i = +1 \text{ for } i = 1, 2, ..., N \\ w^T x_i + b \leq +1 \text{ if } y_i = -1 \text{ for } i = 1, 2, ..., N \end{aligned} \tag{10}$$

Many datasets cannot be separated linearly. Hence, there is no way to satisfy all the constraints in Eq. 10. Therefore, slack variables (ξ_i) are introduced to loosen some constraints in such datasets and still construct useful classifiers. In general, these variables are used for the optimization problem in two ways. First, they help to handle the degree to which the constraint on the i-th datapoint can be violated. Second, by adding the slack variable to the energy function, it aims to simultaneously minimize the use of the slack variables. The mathematical optimization problem formula can be modified as:

$$\min_{w,b,\xi_{i:N}} \sum_i \xi_i + \lambda \frac{1}{2} \|w\|^2 \tag{11}$$

such that, for all i,

$$y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \tag{12}$$

The slack variables are denoted as ξ, with $\xi_i > 1$ for misclassified points and $0 < \xi_i \leq 1$ for points close to the decision boundary, which is a margin violation.

In addition, the Lagrangian (L) is also used for transforming the SVM problem in a manner that is conducive to powerful generalization. In this case, it assumes that the dataset is linearly separable, and so the slack variables are dropped. The Langrangian enables us to re-express the constrained optimization problem (shown as Eq. 10) as an unconstrained problem. Finally, when the Lagrangian is introduced, the SVM objective function shown as Eq. 10 with Lagrange multipliers $\alpha_i > 0$, then becomes:

$$L(w, b, \alpha_{i:N}) = \frac{1}{2}\|w\|^2 - \sum_i \alpha_i(y_i(w^T \phi(x_i) + b) - 1) \tag{13}$$

Consider Eq. 13. The minus sign is used for the second term because this must be minimized with respect to the first term but maximize with respect to the second. Using these constraints on the solution, w becomes:

$$w = \sum_i \alpha_i y_i \phi(x_i), \text{ where } \sum_i y_i \alpha_i = 0 \tag{14}$$

Afterwards, we can replace w (shown as Eq. 14) in Eq. 13. Then, the next constraint, called dual Lagrangian, is applied. The following modified formula is shown as Eq. 15.

$$L(\alpha_{i:N}) = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j k(x_i, x_j) \tag{15}$$

where $k(x_i, x_j)$ is a kernel function that is used to perform non-linear mapping of feature space in which the training set will be classified. Selection of a suitable kernel function is very important for SVM classification performance. Therefore, different SVM algorithms may require diverse types of kernel functions (Figs. 2, 3, 4).

However, the number of examples in each class can be different. Some classes can be sampled very sparsely or even be totally absent. A different number of examples in each class makes it very difficult to use the existing samples to create a model of binary classes prediction. Consequently, one-class SVM (OC-SVM) algorithms were proposed. A classifier model based on OC-SVM is trained on data that has only one class, called the target class, while the other class that may be very sparsely sampled or even entirely absent is called the outlier class. The characteristic of OC-SVM can be useful for anomaly detection since the insufficiency of training examples may characterize these anomalies.

Two well-known OC-SVM algorithms are Schölkopf methodology [27] and support vector data description (SVDD) [28]. The Schölkopf methodology separates all the data points from the origin (in feature space) and maximizes the distance from the origin to the hyperplane, while SVDD assumes a spherical boundary in feature space around the data and reduce the effect of incorporating outliers in the solution by minimizing the volume of the hypersphere.

4.3.2 Schölkopf methodology

The Schölkopf methodology is used to adapt the original SVM to a one-class classification problem [27]. Essentially, after transforming the feature using a kernel, the origin is treated as the only member of the second class. Then, data of the one class are separated from the origin using "*relaxation parameters*".

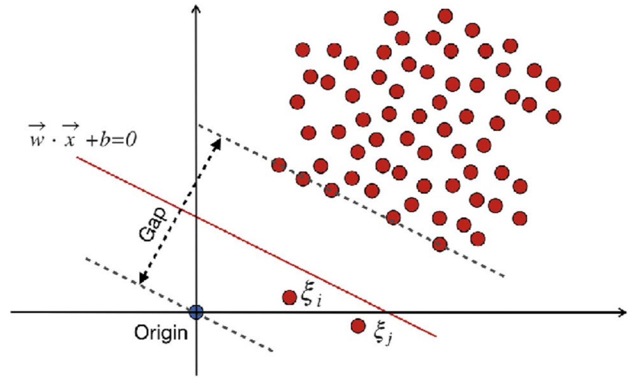

Fig. 3 Overview of Schölkopf methodology

Let $x_1, x_2, , x_l$ be bug reports used as a training set belonging to one class X, where X is a compact subset of $\Re^N$, while $\phi : X \to H$ is a kernel map used to transform the training set to another space. Then, the following quadratic programming problem is solved to separate the data set from the origin.

$$\min \frac{1}{2}\|w\|^2 + \frac{1}{\nu l} \sum_{i=1}^{l} \xi_i - \rho \tag{16}$$

Subject to:

$$(w \times \phi(x_i)) \geq \rho - \xi_i, \text{ where } i = 1, 2, ...N \text{ and } \xi_i \geq 0 \tag{17}$$

When w and ρ are used to solve this problem, the decision function will be positive for most examples of x_i found in the training set of bug reports.

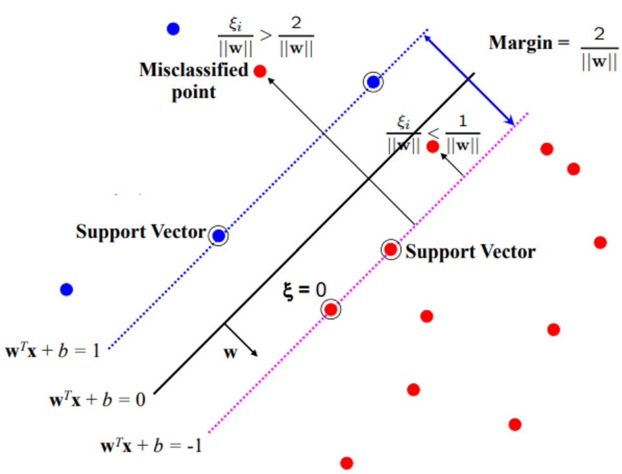

Fig. 2 Overview of traditional binary-class SVM

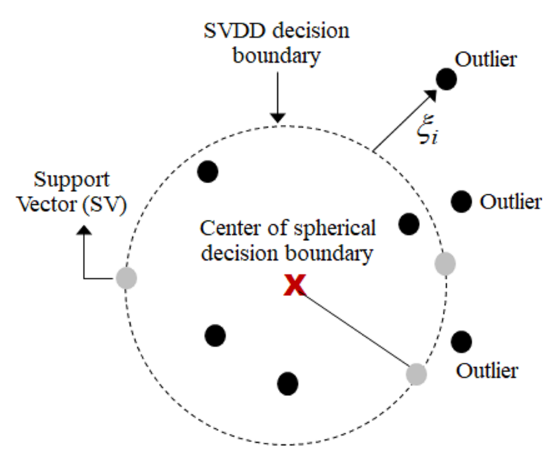

Fig. 4 Overview of SVDD

4.3.3 Support vector data description (SVDD)

SVDD relies on the identification of the smallest hypersphere consisting of all data points, with r and c denoted as radius and center, respectively [28]. Mathematically, the problem can be expressed by following constrained optimization form.

$$\min_{r,c} r^2 \tag{18}$$

Subject to:

$$\|\phi(x_i) - c\|^2 \leq r^2 + \xi_i, \text{ for all } i = 1, 2, ..., l \tag{19}$$

Consider above formulation. It is highly restrictive and sensitive to the presence of outliers. Therefore, a flexible formulation that allows for the presence of outliers is formulated as follows.

$$\min_{r,c} r^2 + \frac{1}{\nu l} \sum_{i=1}^{l} \xi_i \tag{20}$$

Subject to:

$$\|\phi(x_i) - c\|^2 \leq r^2 + \xi_i, \text{ for all } i = 1, 2, ..., l \tag{21}$$

Later, by using the optimality conditions of the Karush-Kuhn-Tucker (KKT), this can be defined as:

$$c = \sum_{i=1}^{l} \alpha_i \phi(x_i) \tag{22}$$

where α_i is the solution to the following optimization problem:

$$\max_{\alpha} \sum_{i=1}^{l} \alpha_i k(x_i, x_j) - \sum_{i,j}^{l} \alpha_i \alpha_j k(x_i, x_j) \tag{23}$$

Subject to:

$$\sum_{i=1}^{l} \alpha_i = 1 \text{ and } 0 \leq \alpha_i \leq \frac{1}{\nu l} \text{ for all } i = 1, 2, ..., l \tag{24}$$

Then, the kernel function provides additional flexibility to the OC-SVM algorithm. Then, this work applies.

This work applies the linear kernel function, where it works well with linearly separable data and most of the text classification problems are linearly separable. This kernel function is faster. In addition, it may be good when there is a lot of features. Definitely, text may have a lot of features.

5 Results and discussion

5.1 The experimental results

This study conducts experiments using two datasets as Herzig's dataset and a real-world dataset relating to Firefox. Experimental results are presented as recall (R) [29], precision (P) [29], and F1 [29] values over different datasets with methods based on the SVM algorithm in Tables 3 and 4.

Table 3 presents experimental results using Herzig's dataset. We applied a feature selection algorithm (i.e. information gain) to select subsets of the features (words) with numbers of 25, 50, 75 and 100. Finally, we trained the classification models as non-bug report identifiers with the SVM family (i.e. binary-class SVM, Schölkopf methodology, and SVDD) before evaluating the performance of each model. Table 3 shows performance comparisons among the non-bug report identifiers. The performance of non-bug report identifiers with *tf-igm* and modified *tf-icf* weighting schemes for both Schölkopf methodology and SVDD methods slightly outperformed when compared to others, while when using 100 features, the performance of all non-bug report identifiers reduced, and using 75 features gave better results than using 25, 50, and 100 features.

In Table 3, using 75 features for non-bug report identifier modeling based on binary-class SVM improved average scores of F1 compared to 25, 50, and 100 features by 5.29%, while using 75 features for non-bug report identifier modeling based on Schölkopf methodology improved average scores of F1 compared to 25, 50, and 100 features by 4.82%. Finally, using 75 features for non-bug report identifier modeling based on SVDD improved average scores of F1 compared to 25, 50, and 100 features by 4.71%. Thus, using 75 features may be suitable for in this study when working on the Herzig's dataset.

Consider Table 4. The numbers of features used are 75, 150, and 225, with results similar to those in Table 3. Non-bug report identifiers performance performed with *tf-igm* and modified *tf-icf* weighting schemes for both Schölkopf methodology and SVDD yielded higher values when comparing to the others, while using 150 features gave better results than using 75 and 225 features.

In Table 4, using 150 features for non-bug report identifier modeling based on binary-class SVM, improved average scores of F1 compared to 75 and 225 features by 4.25%, while using 75 features for non-bug report identifier modeling based on Schölkopf methodology improved average scores of F1 compared to 75 and 225 features by 3.05%. Finally, using 75 features for non-bug report identifier modeling based on SVDD improved average scores of

Table 3 Experimental results using Herzig's dataset

Number of Features used	Term weighting schemes	Binary-class SVM (%)			Schölkopf (%)			SVDD (%)		
		R	P	F1	R	P	F1	R	P	F1
25	tf	69	67	68	71	67	69	71	67	69
	tf-idf	68	65	66	70	65	67	70	65	67
	tf-igm	74	73	73	73	72	72	73	72	72
	Original tf-icf	71	67	69	71	68	69	71	67	69
	Modified tf-icf	74	73	73	73	72	72	73	72	72
50	tf	71	69	70	72	70	71	72	70	71
	tf-idf	70	68	69	72	69	70	72	70	71
	tf-igm	76	75	75	78	75	76	78	76	77
	Original tf-icf	74	72	73	75	73	74	75	74	74
	Modified tf-icf	76	75	75	78	75	76	78	76	77
75	tf	73	71	72	73	71	72	73	71	72
	tf-idf	72	70	71	72	70	71	72	70	71
	tf-igm	**78**	**75**	**76**	**80**	**75**	**77**	**80**	**76**	**78**
	Original tf-icf	75	73	74	75	73	74	75	73	74
	Modified tf-icf	**79**	**76**	**77**	**80**	**76**	**78**	**80**	**76**	**78**
100	tf	68	66	67	71	67	69	71	67	69
	tf-idf	67	64	65	69	66	67	69	66	67
	tf-igm	73	70	71	73	71	72	73	71	72
	Original tf-icf	70	69	69	71	68	69	72	68	70
	Modified tf-icf	73	71	72	73	71	72	73	71	72

Table 4 Experimental results using Firefox dataset

Number of features used	Term weighting schemes	Binary-class SVM (%)			Schölkopf (%)			SVDD (%)		
		R	P	F1	R	P	F1	R	P	F1
75	tf	72	69	70	75	72	73	75	72	73
	tf-idf	71	69	70	75	71	73	75	71	73
	tf-igm	75	72	73	77	74	75	77	75	76
	Original tf-icf	74	71	72	75	72	73	76	73	74
	Modified tf-icf	76	72	74	78	74	76	77	74	75
150	tf	76	74	75	79	74	76	78	74	76
	tf-idf	76	72	72	79	74	76	79	75	77
	tf-igm	**82**	**78**	**79**	**84**	**83**	**83**	**85**	**83**	**84**
	Original tf-icf	79	77	78	80	79	73	79	78	78
	Modified tf-icf	**82**	**79**	**79**	**85**	**82**	**83**	**86**	**82**	**84**
225	tf	75	73	74	77	75	76	77	75	76
	tf-idf	75	72	73	77	74	75	77	74	75
	tf-igm	79	76	77	81	79	80	81	79	80
	Original tf-icf	76	74	75	79	78	78	79	78	78
	Modified tf-icf	79	76	77	81	79	80	81	79	80

F1 compared to 75 and 225 features by 4.61%. Thus, using 150 features may be suitable for this study when working on the Firefox dataset.

Results in Tables 3 and 4 show that the first experiments returned the best results using 75 features (Table 3), while the second experiments return the best results using 150 features (Table 4). This occurred because IG was applied to select the most suitable features. Using this technique helps to select a subset of the most relevant features for the bug report dataset. Consequently, fewer features allow machine learning algorithms such as SVM to run more efficiently and more effectively because this algorithm is sensitive to irrelevant input features, resulting in reduced predictive performance.

5.2 Comparison of the best proposed model against two baselines

We compared the best models of non-bug report identifiers based on the proposed method against two baselines proposed by Pingclasai et al. [13] and Terdchanakul et al. [15]. Then, we compared all methods under the same environmental setting. The experimental results are shown in Table 5.

When using Herzig's dataset, our non-bug report identifier was better than the results from the method proposed by [13] but gave the slightly lower results than the results from the method proposed by [15]. However, surprisingly, when experimenting with a bug report dataset related to Mozilla Firefox, as a real-world dataset, our model based on the proposed method returned better results than the baseline methods with improved scores of F1 at 9.09% for [13] and 6.33% for [15].

5.3 Discussion

Consider the results shown in Tables 3 and 4. All results were satisfactory, although different methods were used. Three main points are discussed as follows.

First, using CamelCase together with unigram should improve the search and may help to increase the scores of recall, precision and F1 because CamelCase words indicate the specificity of the software. Using CamelCase along with unigram keywords may help to increase search efficiency [22]. However, some bug reports contain slang as a version of the language that depicts informal conversation or text that has a different meaning. These words can cause problems during the execution of pre-processing steps and affect the accuracy and efficiency of the text analysis domain. It would be better if these words are converted to formal language in the pre-processing stage before the subsequent processing steps. This point may require consideration in future studies.

Second, when considering term-word weighting schemes, *tf* and *tf-idf* returned satisfactory results but these were lower when compared with *tf-igm*, original *tf-icf*, and modified *tf-icf*. This occurred because the rareness of a term is not considered for *tf*, and rare words may not show as important in a specific class for *tf-idf*. As a result, these rare words may sometimes be overlooked during training and predicting bug reports. By contrast, *tf-igm* and modified *tf-icf* returned the most satisfactory results because these schemes measure the class distinguishing power of a term by combining term frequency with *igm* and *icf* measures, respectively. These schemes may be able to indicate differences of word scores for words in disparate classes. Therefore, *tf-igm* and modified *tf-icf* may return better results than *tf* and *tf-idf*. Similarly, *tf-icf* improves the efficiency of *tf-idf* by not overlooking rare words in each class because the original *tf-icf* represents the level of those words, although rare words occur in a few documents. As a result, the results of *tf-icf* are better than *tf* and *tf-idf*.

Third, this study applied the SVM family i.e. binary-class SVM, Schölkopf methodology and SVDD as the main mechanisms for modeling non-bug report identifiers. Results in Tables 3, 4 and 5 show that non-bug report identifiers based on these algorithms are acceptable compared to [13, 15]. However, results of binary-class SVM were lower than Schölkopf methodology and SVDD because of a class imbalance. Although the same number of documents was used in each class, the number of features in each class may not be the same, and this may reduce classification performance.

In addition, when considering the results shown in Table 5, our proposed model returned the slightly better results than the baseline methods if looking at the overall picture. The reasons for this performance are described above.

6 Conclusions

Bug reports offer important information for improving software quality. To facilitate the collection of large bug reports from more users, many bug tracking systems (BTS) have been proposed and developed. These systems allow users around the world to report, describe, track, classify and comment on their bug reports. Unfortunately, non-bug reports can also be submitted. Therefore, a process of filtering non-bug reports is required. In general, this task is performed manually by bug triagers who are software development experts. However, this process is time-consuming and errors

Table 5 Comparison of the best-proposed model against two baselines

Method	Herzig's dataset			Firefox dataset		
	R	P	F1	R	P	F1
Pingclasai et al. [13]	76	78	77	76	78	77
Terdchanakul et al. [15]	78	80	79	78	80	79
The proposed method based on SVDD with tf-igm term weighting	77	80	78	85	83	84

in bug report analysis often occur. Thus, the challenge here was to present a method of automatically filtering non-bug reports from the BTS. The outcomes are summarized as follows. Firstly, unigram and CamelCase may be suitable for bug report studies. Unigram words are easy to generate and can be found in any bug report, while CamelCase words indicate the specificity of the software. Secondly, the *tf-igm* and *tf-icf* family are supervised weighting schemes that give better results than *tf* and *tf-idf* because they indicate different scores for words in different classes. Simply speaking, they can measure the class distinguishing power of a term and this helps to increase the classification performance. Finally, the SVM family works well for this problem. OC-SVM algorithms may be better than binary-class SVM that often face a problem of class imbalance that reduces classification performance. Our results proved acceptable compared with well-known base-line studies. The performance of non-bug report identifiers with *tf-igm* and modified *tf-icf* weighting schemes for both Schölkopf methodology and SVDD methods yielded the best values compared to other methods.

Furthermore, we also selected the best models of non-bug report identifiers based on the proposed method and used these models to compare with the two baselines proposed by Pingclasai et al. [13] and Terdchanakul et al. [15].

Results show that our method improved F1 scores over the baseline by 9.09% for [13] and 6.33% for [15] when experimenting on the open-source bug report dataset related to Mozilla Firefox. However, when using Herzig's dataset, our model performed better than the method proposed by [13] but gave slightly lower results than achieved by [15]. When looking at the overall picture, our proposed model returned slightly better results than the baseline methods for the reasons mentioned earlier. Findings demonstrate that our proposed method may improve the chances of obtaining better performance for non-bug report identification. Therefore, our proposed method is a good option for non-bug report identification.

However, no method can work well with every dataset; therefore, and we cannot guarantee that our proposed method will work well for other datasets.

Acknowledgements This research was Financially Supported by the Faculty of Informatics, Mahasarakham University (Grant year 2021).

References

1. Sandusky RJ, Gasser L, Ripoche G (2004) Bug report networks: Varieties, strategies, and impacts in a f/oss development community. In: MSR, pp. 80–84. IET
2. Jalbert N, Weimer W (2008) Automated duplicate detection for bug tracking systems. In: 2008 IEEE international conference on dependable systems and networks with FTCS and DCC (DSN), pp. 52–61. IEEE (2008)
3. Wang X, Zhang L, Xie T, Anvik J, Sun J (2008) An approach to detecting duplicate bug reports using natural language and execution information. In: Proceedings of the 30th international conference on Software engineering, pp. 461–470 (2008)
4. Bhattacharya P, Neamtiu I (2011) Bug-fix time prediction models: can we do better? In: Proceedings of the 8th working conference on mining software repositories, pp. 207–210 (2011)
5. Tian Y, Sun C, Lo D (2012) Improved duplicate bug report identification. In: 2012 16th European conference on software maintenance and reengineering, pp. 385–390. IEEE (2012)
6. Zhang J, Wang X, Hao D, Xie B, Zhang L, Mei H (2015) A survey on bug-report analysis. Sci China Inform Sci 58(2):1–24
7. Aggarwal K, Timbers F, Rutgers T, Hindle A, Stroulia E, Greiner R (2017) Detecting duplicate bug reports with software engineering domain knowledge. J Softw 29(3):e1821
8. Anvik J, Murphy GC (2011) Reducing the effort of bug report triage: recommenders for development-oriented decisions. ACM Trans Softw Eng Methodol (TOSEM) 20(3):1–35
9. Luaphol B, Srikudkao B, Polpinij J, Kaenampornpan M (2018) Assembling relevant bug report using the constraint-based k-means clustering. In: 2018 International conference on information technology (InCIT), pp. 1–6. IEEE (2018)
10. Antoniol G, Ayari K, Di Penta M, Khomh F, Guéhéneuc YG (2008) Is it a bug or an enhancement? a text-based approach to classify change requests. In: Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds, pp. 304–318 (2008)
11. Runeson P, Alexandersson M, Nyholm O (2007) Detection of duplicate defect reports using natural language processing. In: 29th international conference on software engineering (ICSE'07), pp. 499–510. IEEE (2007)
12. Herzig, K., Just, S., Zeller, A.: It's not a bug, it's a feature: how misclassification impacts bug prediction. In: 2013 35th international conference on software engineering (ICSE), pp. 392–401. IEEE (2013)
13. Pingclasai, N., Hata, H., Matsumoto, K.i.: Classifying bug reports to bugs and other requests using topic modeling. In: 2013 20Th asia-pacific software engineering conference (APSEC), vol. 2, pp 13–18. IEEE (2013)
14. Limsettho, N., Hata, H., Monden, A., Matsumoto, K.: Automatic unsupervised bug report categorization. In: 2014 6th international workshop on empirical software engineering in practice, pp. 7–12. IEEE (2014)
15. Terdchanakul, P., Hata, H., Phannachitta, P., Matsumoto, K.: Bug or not? bug report classification using n-gram idf. In: 2017 IEEE international conference on software maintenance and evolution (ICSME), pp. 534–538. IEEE (2017)
16. Qin, H., Sun, X.: Classifying bug reports into bugs and nonbugs using lstm. In: Proceedings of the tenth Asia-Pacific symposium on internetware, pp. 1–4 (2018)
17. Pandey, N., Hudait, A., Sanyal, D.K., Sen, A.: Automated classification of issue reports from a software issue tracker. In: Progress in Intelligent Computing Techniques: Theory, Practice, and Applications, pp. 423–430. Springer (2018)
18. Zhou Y, Tong Y, Gu R, Gall H (2016) Combining text mining and data mining for bug report classification. J Softw 28(3):150–176
19. Almhana, R., Mkaouer, W., Kessentini, M., Ouni, A.: Recommending relevant classes for bug reports using multi-objective search. In: 2016 31st IEEE/ACM International conference on automated software engineering (ASE), pp. 286–295. IEEE (2016)
20. Zhou, J., Zhang, H., Lo, D.: Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In: 2012 34th International conference on software engineering (ICSE), pp. 14–24. IEEE (2012)

21. Ye X, Bunescu R, Liu C (2015) Mapping bug reports to relevant files: A ranking model, a fine-grained benchmark, and feature evaluation. IEEE Trans Softw Eng 42(4):379–402
22. Wang, H., Liang, Y., Fu, L., Xue, G.R., Yu, Y.: Efficient query expansion for advertisement search. In: Proceedings of the 32nd international ACM SIGIR conference on research and development in information retrieval, pp. 51–58 (2009)
23. Aizawa A (2003) An information-theoretic perspective of tf-idf measures. Inform Process Manag 39(1):45–65
24. Chen K, Zhang Z, Long J, Zhang H (2016) Turning from tf-idf to tf-igm for term weighting in text classification. Expert Syst Appl 66:245–260
25. Lertnattee, V., Theeramunkong, T.: Analysis of inverse class frequency in centroid-based text classification. In: IEEE International symposium on communications and information technology, 2004. ISCIT 2004., vol. 2, pp. 1171–1176. IEEE (2004)
26. Joachims, T.: Text categorization with support vector machines: Learning with many relevant features. In: European conference on machine learning, pp. 137–142. Springer (1998)
27. Manevitz LM, Yousef M (2001) One-class svms for document classification. J Mach Learn Res 2:139–154
28. Tax DM, Duin RP (2004) Support vector data description. Mach Learn 54(1):45–66
29. Baeza-Yates, R., Ribeiro-Neto, B., et al.: Modern information retrieval, vol. 463. ACM press New York (1999)