Information and Software Technology 147 (2022) 106906

Contents lists available at ScienceDirect

Information and Software Technology

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

CASMS: Combining clustering with attention semantic model for identifying security bug reports

Xiaoxue Ma [a], Jacky Keung [a], Zhen Yang [a], Xiao Yu [b,c,*], Yishu Li [a], Hao Zhang [a]

[a] Department of Computer Science, City University of Hong Kong, Hong Kong, China
[b] School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, China
[c] Wuhan University of Technology Chongqing Research Institute, Chongqing, China

ARTICLE INFO

Keywords:
Security bug report
Clustering
Hybrid neural networks

ABSTRACT

Context: Inappropriate public disclosure of security bug reports (SBRs) is likely to attract malicious attackers to invade software systems; hence being able to detect SBRs has become increasingly important for software maintenance. Due to the class imbalance problem that the number of non-security bug reports (NSBRs) exceeds the number of SBRs, insufficient training information, and weak performance robustness, the existing techniques for identifying SBRs are still less than desirable.
Objective: This prompted us to overcome the challenges of the most advanced SBR detection methods.
Method: In this work, we propose the CASMS approach to efficiently alleviate the imbalance problem and predict bug reports. CASMS first converts bug reports into weighted word embeddings based on $tf-idf$ and $word2vec$ techniques. Unlike the previous studies selecting the NSBRs that are the most dissimilar to SBRs, CASMS then automatically finds a certain number of diverse NSBRs via the Elbow method and k-means clustering algorithm. Finally, the selected NSBRs and all SBRs train an effective Attention CNN–BLSTM model to extract contextual and sequential information.
Results: The experimental results have shown that CASMS is superior to the three baselines (i.e., FARSEC, SMOTUNED, and LTRWES) in assessing the overall performance (g-measure) and correctly identifying SBRs (*recall*), with improvements of 4.09%–24.26% and 10.33%–36.24%, respectively. The best results are easily obtained under the limited ratio ranges of the two-class training set (1:1 to 3:1), with around 20 experiments for each project. By evaluating the robustness of CASMS via the standard deviation indicator, CASMS is more stable than LTRWES.
Conclusion: Overall, CASMS can alleviate the data imbalance problem and extract more semantic information to improve performance and robustness. Therefore, CASMS is recommended as a practical approach for identifying SBRs.

1. Introduction

Nowadays, bug tracking systems are commonly used to help developers maintain software products. It is significant for security engineers to accurately capture software bugs in the bug reports that users have submitted. Such reports can be typically divided into two categories: Security Bug Reports (SBRs) and Non-Security Bug Reports (NSBRs). Generally, SBRs prioritize being detected, and those security vulnerabilities exposed by SBRs should be fixed before they are disclosed [1]. If a security bug is published, it may cause potential security breaches to systems and significant damage to the software products [2]. However, it is not feasible to manually identify SBRs in massive bug reports due to the lack of security-related professional knowledge and the high cost of time. To identify SBRs more efficiently and accurately, text-based prediction methods have been proposed to predict whether or not new bug reports are security-related. Some of these methods [2–4] first represent bug reports as feature vectors by identifying relevant keywords and extracting feature information based on term frequency calculation, such as term-by-document frequency matrix [2] and term frequency–inverse document frequency ($tf-idf$) [3,4]. Then, these methods employ some classification algorithms (e.g., Naïve Bayes, Logistic Regression, etc.) to construct a prediction model based on the feature vectors.

* Corresponding author at: School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, China.
E-mail addresses: xiaoxuema3-c@my.cityu.edu.hk (X. Ma), jacky.keung@cityu.edu.hk (J. Keung), zhyang8-c@my.cityu.edu.hk (Z. Yang), xiaoyu@whut.edu.cn (X. Yu), yishuli5-c@my.cityu.edu.hk (Y. Li), hzhang339-c@my.cityu.edu.hk (H. Zhang).

https://doi.org/10.1016/j.infsof.2022.106906
Received 22 June 2021; Received in revised form 9 February 2022; Accepted 16 March 2022
Available online 26 March 2022

The challenges of identifying SBRs usually include:

(1) How to alleviate the class imbalance problem of the training dataset.

Due to the nature of the collected datasets, there is an unavoidable problem that the percentage of SBRs in datasets only ranges from 0.5% to 9% [3,5,6]. The prediction models trained on the highly imbalanced datasets will focus more on the NSBRs and predict unknown bug reports such as NSBRs. However, security engineers are willing to detect SBRs more accurately. To alleviate the data imbalance problem, most recent methods [3,5,6] first rank NSBRs according to some metrics (e.g., keyword scoring [3,5] and $BM25F_{ext}$ for similarity calculation [6]). Peters et al. [3] selected the top 100 terms in SBRs with the highest $tf-idf$ values as security-related keywords and found that at least 74% of the security-related keywords also appeared in the context of NSBRs, if each keyword is considered a feature, most NSBRs have features that overlap with SBRs. Nevertheless, Jiang et al. [6] pointed out that most of the security-related keywords selected by Peters et al. [3] may not be related to security. Thus, they ranked NSBRs based on content similarity instead of the individual word. By calculating the distance between bug reports (e.g., Euclidean distance and Minkowski distance), they evaluate the similarity between them. The larger the distance, the more dissimilar the two bug reports. Then, these methods only retain the NSBRs that is the furthest to all SBRs in the feature space, i.e., the most dissimilar ones.

(2) How to capture the semantic and sequential information of bug reports better.

Most of the previous studies [2,3,5] adopted term-based approaches (e.g., $tf-idf$) to evaluate the importance of each term in bug reports. They applied machine learning algorithms to build classifiers, which fail to capture semantic information and the long-term dependency in sentences. A bug report contains 75.86 terms on average in our experiment datasets, and the longest one has 4859 terms. A lot more information is required to train a prediction model, such as the order and the semantic information of terms. In contrast, insufficient information will cause high-dimensional sparsity in the feature space and the loss of temporal relationships between terms.

(3) How to make the approach more efficient and robust.

Jiang et al.'s method [6] achieve good performance, which requires many repeated experiments (i.e., perform a total of 240 experiments on each project), and the ratio of NSBRs to SBRs of the best result on each project ranges from 1:1 to 10:1 under the 12 combinations of two filters and six classifiers. In addition, the indicator for evaluating the robustness of the approach is not ideal, i.e., the g-measure of LTRWES varies significantly with the resampling ratio. Hence, its g-measure has a high standard deviation.

Considering the above challenges, we propose a novel approach called CASMS (combining Clustering with Attention Semantic Model for identifying Security bug reports) to efficiently alleviate the imbalance problem in bug report datasets and improve the overall performance of bug report classification with good stability. Since it is impossible to distinguish the two categories only based on the feature distance, unlike previous approaches that select the most dissimilar NSBRs to SBRs, we adopted the k-means clustering algorithm to implicitly represent a partition such that it can self-organize data groups according to the original data structure, which divides all NSBRs into several clusters and select those samples closest to the centroids from each cluster. By this means, gathering NSBRs from each cluster is able to encompass different potential types of NSBRs to extract diverse NSBRs. Specifically, we first use the Elbow algorithm to calculate the optimal number of clusters k. Then a certain number of samples closest to the centroid in each cluster are identified. In this way, we can ensure that the selected NSBRs are sufficiently diverse.

After that, the retained NSBRs and all SBRs are used for training. The essence of security bug report detection is a text-based analysis task, and a model based on CNN–BLSTM architecture is applied for prediction. The retained data is first processed by the Convolutional Neural Networks (CNN) layer to capture the correlation between consecutive terms in bug reports, and then the BLSTM (Bidirectional Long Short-Term Memory) layer is used to extract the correlation between long-distance terms. Followed by the Attention layer, it pays more attention to the relevant part of the input. In comparison, the prediction model takes contextual and sequential information of bug reports into consideration, while the general machine learning algorithms adopted by prior studies [2,3,5,6] cannot.

Although CASMS is 4.83%–14.99% inferior to the three baselines (i.e., FARSEC [3], SMOTUNED [5], and LTRWES [6]) in terms of the average false alarm rate (pf) shown in our experimental results, CASMS outperforms the three baselines by 4.09%–24.26% in terms of the average g-measure as a comprehensive metric for evaluating the models, 10.33%–36.24% in terms of the average *recall*. In addition, the results demonstrate that the ratio range of the training NSBRs to SBRs for the optimal results is narrowed (between 1:1 and 3:1), and the average standard deviation of g-measure of CASMS under several resampling ratios is 1.47%–7.4% lower than that of LTRWES. For instance, for the *Wicket* project, the standard deviation of CASMS is 7.44%, and the standard deviation of LTRWES with rs-filter and ms-filter are 11.74% and 15.49%, respectively.

In summary, this paper makes the following contributions:

- We propose a novel SBRs detection approach called CASMS, which can automatically extract diverse NSBRs and learn the semantic information of bug reports to improve the performance.
- Empirical studies on five real-world projects show that CASMS is more efficient and robust without much tuning efforts by automatically finding the optimal number of clusters k^* and narrowing the ratio scope of NSBRs to SBRs.

The organization of the remaining sections is as follows. Section 2 introduces the related work and background. Section 3 presents the framework of CASMS and the details. Sections 4 and 5 present the experimental setup and experiments results, respectively. Section 6 discusses the threats to validity. Finally, we conclude our work in Section 7.

2. Related work and background

2.1. SBR detection

In the early phase, bug tracking systems are used to help engineers to identify duplicate bug reports [7–11], evaluate the severity or priority of bug reports [12–14], trace bug reports back to relevant source documents [15–19], analyze and predict the effort needed to fix software bugs [20], work on characteristics of software vulnerabilities [21,22], and evaluate the ability of code analysis tools to detect security vulnerabilities [23]. Most of these methods applied textual similarity metrics (e.g., cosine similarity) and machine learning methods (e.g., SVM and KNN) to extract textual information, while the sequential and semantic information have not been considered. In addition, there are limited studies related to security bug report detection [2–6], which only applied machine learning methods for model training. Therefore, the sequential and semantic information have not been considered in these previous studies.

Gegick et al. [2] for the first time raised a problem that bug reporters are likely to mislabel SBRs as NSBRs due to the lack of security domain knowledge. In order to identify those SBRs that have been manually mislabeled as NSBRs, they applied text mining on textual descriptions of bug reports to generate term-by-document frequency matrices, and then trained a statistical model based on the matrices. Their empirical results show that a high percentage (78%) of SBRs are mislabeled as NSBRs on a large Cisco software system, which indicates that these extracted features (i.e., the security-related keywords) used for training may cause misclassification of bug reports.

Peters et al. [3] have found that at least 74% of the security-related keywords appearing in both SBRs and NSBRs, that are considered as security cross words, may result in mislabeling SBRs. Therefore, they proposed a framework FARSEC to detect SBRs by filtering NSBRs with security cross words. They first calculated the $tf-idf$ value of each term and selected the top-100 terms with the highest $tf-idf$ values as security-related keywords, and then used the terms to build term-document matrices (also called feature set). After that, each term in the training set was scored, and terms appearing in the feature set were given higher weights via different filters. Finally, all bug reports were ranked, and the NSBRs whose scores were higher than a threshold were filtered. Additionally, they adopted CLNI (Closet List Noise Identification) [24], a noise detection algorithm based on Euclidean Distance, for prediction together.

Goseva et al. [4] proposed a supervised approach and an unsupervised approach to identify security bug reports. For both approaches, they adopted three methods to extract feature vectors, i.e., Binary Bag-of-Words Frequency (bf), Term Frequency (tf), and Term Frequency–Inverse Document Frequency ($tf-idf$). For the supervised approach, they combined machine learning algorithms with feature vectors to classify bug reports. The difference from other methods is that they attempted different-size training set to determine the smallest size that can achieve good classification results. Furthermore, they proposed an unsupervised approach based on the concept of anomaly detection to identify the security bug reports. Two NASA missions from issue tracking systems are used to evaluate the proposed two approaches. However, our study aims to alleviate class imbalance problem and identify security bug reports more efficiently, which is not consistent with their focus (e.g., determine the smallest size of training set). In addition, the dataset they used is far less serious than the class imbalance problem of our dataset. Thus, this method is not considered as one of our baselines.

On the basis of FARSEC, Shu et al. [5] combined SMOTE [25] with the filters proposed by FARSEC to preprocess the data. They adopted the Differential Evolution (DE) algorithm [26,27] to obtain the optimal parameters of SMOTE and the learners (i.e., the machine learning classifiers) separately. According to their experiments, they recommended using DE just for SMOTE, called SMOTUNED, rather than learners due to the time-consuming tuning. In the end, the recall (pd) of the prediction models has been significantly improved with a moderate increment of false alarm rate (pf), so that SMOTUNED outperforms FARSEC in terms of g-measure.

In another recent work, Jiang et al. [6] put forward LTRWES. Unlike keyword-based FARSEC, LTRWES was a content-based data filtering approach, which considered the content of bug reports rather than individual keywords. It first calculated the content similarity between each NSBR and all SBRs by using a ranking model $BM25F_{ext}$ [28], and then extracted top-m NSBRs that were the most irrelevant to SBRs. The idea of LTRWES is similar to FARSEC, i.e., retain the most dissimilar NSBRs to SBRs in order to achieve a higher recall rate. To avoid vector sparsity and extract the semantic information, $word2vec$ was also used in LTRWES to transform the remaining NSBRs together with SBRs into low-dimensional vectors before training.

However, there are some problems in the previous methods. (1) These methods retained the most dissimilar NSBRs to SBRs rather than diverse NSBRs. (2) They applied some classification algorithms to train the model, which does not take the semantic and sequential information of bug reports into account. (3) Although LTRWES achieved good performance, it is not robust even though they conducted repeated experiments many times. For example, under the combination of SVM classifier and ms filter, they tried the ratio of NSBRs to SBRs from 1:1 to 10:1 for training to find the optimal prediction results, while the number of the remaining training NSBRs ranges from 4 to 770. In addition, the g-measure of LTRWES varies significantly with the resampling ratio.

Therefore, our approach automatically selects diverse NSBRs via k-means clustering algorithm at the data filtering stage and constructs an Attention CNN–BLSTM model to extract time-series information for training. To avoid tons of repeated trials, we introduce the Elbow algorithm to find the optimal number of clusters and narrow the optimal resampling ratio range of NSBRs to SBRs in the training set.

2.2. Textual representation

Text classification is usually defined as the process of identifying the category of a new document based on the probability suggested by a designated training corpus with labels [29], which is typically viewed as a supervised learning task. Since the classification of bug reports into two categories depends on the interpretation of the text, techniques for text representation are required to facilitate subsequent training.

2.2.1. tf-idf

Term Frequency–Inverse Document Frequency ($tf-idf$) is one of the most popular term-weighting schemes in text analysis and data mining [30,31]. In the field of information retrieval, $tf-idf$ is represented as a numerical statistic value, reflecting how important a term is to a document in a corpus or collection. Terms with higher $tf-idf$ values indicate a powerful ability to distinguish the categories of documents and have a stronger relationship with the documents where these terms appear [29]. Given a document collection D, a term t, and an individual document d that belongs to D. The calculation of $tf-idf$ is as following [30]:

$$t_d = f_{t,d} * log(|D|/f_{t,D}), \quad (1)$$

where $f_{t,d}$ is the number of times that term t appears in the document d, $|D|$ is the size of the corpus, and $f_{t,D}$ is the number of documents in which the term t appears in D.

$tf-idf$ is widely used for some tasks related to bug reports, such as identifying hidden impact bugs [32] and SBRs detection [3,4]. Although $tf-idf$ supposes each term in bug reports is independent, it reflects the importance of the term to bug reports where the term exists to a certain extent. Thus, we apply $tf-idf$ to calculate the weight of terms instead of directly generating word vectors in our proposed approach.

2.2.2. Word embedding

It has already been proved that word embedding is critical to construct deep learning models in many NLP tasks [33–40]. In the early days, the original objective of word embedding was to convert text into a statistical representation, such as the one-hot representation. For each word, its corresponding position was set to 1, and the other positions were all 0. However, the earliest sparse-populated vector was very verbose because it set the dimension of the word vector to be the size of the entire vocabulary, which may cause the dimensional disaster. In addition, the contextual information of words was ignored. To address this problem, Mikolov et al. [41–43] proposed two language learning algorithms, CBOW and Skip-Gram based on $word2vec$, to represent the meaning of a word with respect to other words. The former algorithm predicts the surrounding words according to the given current word, while the latter algorithm predicts the current words based on the distributed representation of the context [29]. Generally, owing to the higher calculation complexity of the Skip-Gram model, CBOW is more suitable for scenarios with small datasets [6,41]. LTRWES, an approach for security bug report detection, used CBOW to convert the bug reports into low-dimensional vectors for further training. Referring to that, we multiply the word vectors and the previously calculated weights of words to get the feature vectors of our datasets.

2.3. Clustering for dealing with data imbalance

Clustering, as an unsupervised machine learning technique in data science, automatically groups similar objects into a single cluster based on certain common characteristics. Within the same cluster, the variance of entities is minimized. It has been extensively studied in machine learning, such as feature selection [44,45], distance algorithms [46], and grouping methods [47]. To identify diverse NSBRs, clustering algorithms can divide data into several categories in terms of data features. Typical approaches are introduced below.

By and large, (1) *hierarchical-based* methods [48] (e.g., CHAMELEON) usually cause high time complexity [49,50]. Due to the high time requirement for this study, they will not be considered in this study; (2) the parameter setting has a great effect on the clustering results of *density-based* methods. For example, DBSCAN [51,52] uses fixed parameters to identify clusters. However, when the sparse degree of clusters is different, the same criterion may harm the natural structure of clustering [52], which is contrary to our view of selecting samples based on the relationship between features; (3) *grid-based* methods [53,54] such as CLIQUE are fast, but they are sensitive to the chosen parameters [55]. Especially the threshold selection, the high threshold may lose the clusters, and the low threshold may merge clusters that should be separated. Besides, they cannot handle irregularly distributed data and are prone to dimensional disasters; (4) *partition-based* methods [56] are efficient and straightforward by referring to [49,50,57]. Although the result is easy to be locally optimal and the number of clusters k needs to be set in advance, combining multiple trials with the Elbow principle [58,59] can make up for this problem.

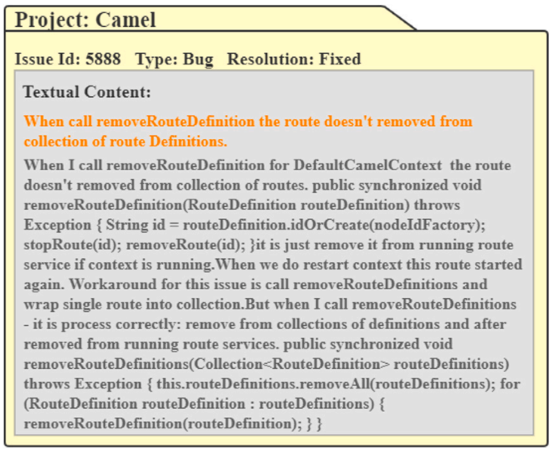
Ultimately, we choose one of the most popular partition-based method k-means [60] for data filtering. According to the incoming data and the set k, k-means constructs the initial centroids and adjusts them in terms of the distance between centroids and samples. The final centroids generated, as well as the distances between each sample and its centroid, are stored.

Before trying multiple k values in turn, the recommended k^* can be roughly calculated based on the Elbow method. k-means is to minimize the square error between the samples and the centroid as the objective function. The sum of the squared distance error between the centroid of each cluster and the sample points in the cluster is called the distortion degree (distortions). For a cluster, the lower the degree of distortion, the tighter the members in the cluster; the higher the degree of distortion, and the looser the structure of the cluster. The degree of distortion will decrease as the number of clusters k increases. However, for data with a certain degree of discrimination, the degree of distortion improves when a certain critical point is reached and then slowly declines. This critical k^* can be considered as the optimal value. To avoid spending much more time trying on the value of k, we use the Elbow method to find the recommended k^*, and then employ k-means algorithm for clustering and filtering NSBRs.

2.4. Prediction models

2.4.1. Machine learning algorithms

After extracting the feature of bug reports, the recent studies [3–6] tend to apply some widely used classification algorithms to build a SBR prediction model, such as Naïve Bayes (NB) [61,62], Logistic Regression (LR) [63,64], Support Vector Machine (SVM) [65], Random Forest (RF) [66], Multilayer Perceptron (MP) [67], and K Nearest Neighbor (KNN) [68]. However, these algorithms cannot keep the information of previous terms continuously, instead throwing everything away and restarting the calculation. In other words, the algorithms cannot capture the sequential information in the input data required to process the following data.


Project: Camel

Issue Id: 5888 Type: Bug Resolution: Fixed

Textual Content:

When call removeRouteDefinition the route doesn't removed from collection of route Definitions.

When I call removeRouteDefinition for DefaultCamelContext the route doesn't removed from collection of routes. public synchronized void removeRouteDefinition(RouteDefinition routeDefinition) throws Exception { String id = routeDefinition.idOrCreate(nodeIdFactory); stopRoute(id); removeRoute(id); }it is just remove it from running route service if context is running.When we do restart context this route started again. Workaround for this issue is call removeRouteDefinitions and wrap single route into collection.But when I call removeRouteDefinitions - it is process correctly: remove from collections of definitions and after removed from running route services. public synchronized void removeRouteDefinitions(Collection<RouteDefinition> routeDefinitions) throws Exception { this.routeDefinitions.removeAll(routeDefinitions); for (RouteDefinition routeDefinition : routeDefinitions) { removeRouteDefinition(routeDefinition); } }

Fig. 1. A SBR instance of the *Camel* project (the text in the orange font represents *summary* and the text in the gray font represents *description*). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

2.4.2. Time series models

Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) are two mainstream architectures for modeling and understanding natural language. CNN performs excellently in learning local responses from temporal data, and RNN that allows information to persist is good at sequential modeling [69]. Recurrent neural networks (RNN) [70] is designed to capture long-term dependency within the sequential data, which adopts a simple mechanism of recurrent feedback. In the field of information retrieval, term dependencies [71] are usually defined as statistical co-incidence of terms on the scale of whole documents. The long-term dependencies can be thought of as the relatively long intervals of relevant information and locations. Long Short-Term Memory (LSTM) architecture is derived from RNN. It has recursive connections so that the previous activation state of neurons from previous time steps is used as the context for forming the output. Bidirectional LSTM (BLSTM) was initially proposed by Graves [72] to avoid gradient exploding and vanishing, and it had been applied in many speech recognition tasks [72,73]. Hence, BLSTM focuses on extracting the correlation between long-distance terms but insufficiently captures the correlation between consecutive terms. For this reason, a mixed model CNN–BLSTM was designed for extracting features and was broadly used in multiple fields [74–77]. Attention mechanism [78] was proposed by the Bengio team to allocate weights to terms. Its purpose is to give components more weight for identifying which components in the input sentence have greater impacts on the classification results. It has been broadly employed in translation, image captioning, and speech recognition [78–80]. Attention mechanism is originated from the information retrieval system. The principle of it is that people enter a sentence (Q) and the search engine will match the keywords (K). Based on the similarity between them, the matching content (V) will be returned.

In this study, bug reports often contain a *summary* and a detailed *description* (an instance demonstrated in Fig. 1). In our datasets, a bug report contains an average of 75.86 terms, a median of 44 terms, and the longest one contains 4859 terms. Since Khandelwal et al. [81] has pointed out that LSTM language models is capable of using context size of 200 tokens on average, the LSTM architecture can be applied to capture the long-dependencies relationship between terms in our datasets where neither the average length nor the median exceeds 200 terms. Therefore, CNN–BLSTM is an optimal choice as our prediction model. To be more rigorous, we adopt the self-Attention mechanism in our prediction model is to automatically discover those terms that play a critical role in classification.

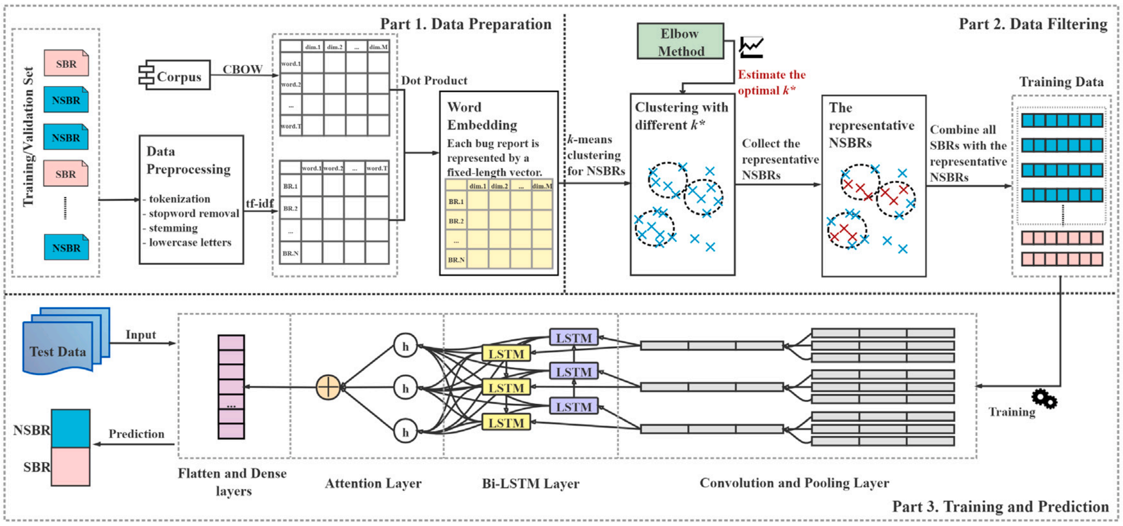


Fig. 2. CASMS Framework.

3. CASMS framework

The CASMS framework shown in Fig. 2 consists of three stages. In the first stage, CASMS converts bug reports into weighted word embeddings based on $tf-idf$ and $word2vec$ techniques. In the second stage, CASMS effectively extracts diverse NSBRs via the Elbow method and k-means clustering algorithm. In the third stage, CASMS trains an effective Attention CNN–BLSTM model using the selected NSBRs and all SBRs. The detailed processes are explained in the following subsections.

3.1. Data preparation

Preprocessing. Word embedding and $tf-idf$ algorithms explained in Algorithm 1 are applied to represent bug reports in this section. Before that, we adopt the following steps to preprocess our data. Firstly, we filter out the noise information by matching it with a set of regular expression rules (e.g., removing URLs, Stack Trace, and hex code). Then, we tokenize the textual information and remove the stop words. In addition, all terms are changed into lowercase, and they are stemmed.

Text representation. In Algorithm 1, CASMS calculates the $tf-idf$ weight of each term in the training set (Lines 1–4) and the vector of each term in the open-sourced corpus of collected bug reports [6] based on $word2vec$ (Lines 5–8). Then, CASMS traverses all the terms in the training set (Line 9). If a term exists in the corpus (Line 10), the algorithm assigns the calculated word vector to this term (Line 11); otherwise, it will be represented by a zero vector of the same length (Line 12–13). Finally, through dot product by weights and vectors, each bug report in the generated matrix can be expressed as a same-length weighted vector (Line 16–17).

Algorithm 1 Calculate the vectors to represent bug reports. $CalVec(S, C, l)$

Input: Training set S, bug report corpus C, the set length of each term l
Output: The vectors of bug reports M_{s2v}

```
1: for each term $w$ in $S$ do
2:     $M_{weight} \leftarrow Transform(tfidf(w))$
3:     // Generate the weight matrix of terms in $S$;
4: end for
5: for each term $t$ in $C$ do
6:     $M_{w2c} \leftarrow Word2Vector(t)$
7:     // Return the matrix consisting of the vectors of each term in the corpus $C$;
8: end for
9: for each term $w$ in $S$ do
10:    if $w$ exists in $C$ then
11:        $M'_{w2c}(w) \leftarrow M_{w2c}(w)$
12:    else
13:        $M'_{w2c}(w) \leftarrow M([0,]*l)$
14:    end if
15: end for
16: $M_{s2v} \leftarrow dot(M_{weight}, M'_{w2c})$
17: return $M_{s2v}$
18: // Return the matrix of bug reports, each row is a vector to represent a bug report.
```

3.2. Data filtering

Rough estimation on k values. As explained in Section 2.3, we choose the widely-used partition-based method k-means as the main algorithm for data filtering. To ensure the efficiency of our approach, before clustering and filtering NSBRs, we first estimate the range K of the number of clusters k and compute the recommended k^* based on the Elbow method (Algorithm 2). For each chosen k, Algorithm 2 calculates the distance between each sample and its corresponding initialized cluster centroid. By summing and normalizing these distances, the average distortion for each k (Line 2) is worked out (Lines 3–5). When the degree of distortion has been greatly improved at a specific point, followed by a gentle improvement, the k value at this point is suggested to be used for next clustering. To make the result more visualized, take the *Derby* project as an example, a line graph is drawn in terms of k values (Line 7–8) and the corresponding distortion illustrated in Fig. 7, the turning point that can be regarded as k^*, which is marked with a dark blue arrow. For each project, we execute Algorithm 2 ten times, and the experimental results are shown in Table 5.

Algorithm 2 Compute the recommended cluster numbers k for NSBRs. **CalClusterNum**(K, S, C, l)

Input: The selected range K, training set S, corpus C, the set length of each term l
Output: The recommended k value

```
1: $senVec \leftarrow CalVec(S, C, l)$
2: for $k$ in $K$ do
3:     classifier $clf_k \leftarrow KMeans(k)$
4:     $clf_k \leftarrow fit(senVec_{nsbr})$
5:     $meanDistortion_k \leftarrow \sum_{n=1}^{S}(\frac{\min(cDist(senVec_{nsbr}, clf_{centroid}))}{S_{nsbr}})$
6: end for
7: $line\ graph \leftarrow Draw(meanDistortion_k)$
8: // Draw a line graph based on the $k$ and its mean distortion value;
9: return $k^*$
10: // Return the $k$ value at the turning point.
```

Clustering. The clustering here is clustering within one class (NSBRs). Because our datasets no longer distinguish and label the NSBRs, we apply the clustering method within the single class to find the potential relationship between features. Algorithm 3 is designed to find the appropriate number of clusters and the suitable total number of selected NSBRs via automatic clustering. Given the range K, for each k in K, the algorithm initializes the centroid for each cluster (Line 3–4) and the zero matrices that include the index of the cluster to which each sample belongs and the distance from each sample to this cluster (Line 5–6). For each sample, the algorithm iteratively calculates and saves the index of its nearest cluster and the distance between it and the centroid of this cluster (Lines 11–21). During this process, if the saved index is the index of the nearest cluster for all samples, the centroids will no longer change (Line 10). Otherwise, the centroids will be changed gradually according to the average distance between the cluster centroid and all samples in this cluster (Lines 22–25). At last, each sample has its cluster index and the distance from the centroid. The NSBR set for succeeding training can be gathered according to different k values and different numbers n of samples in each cluster (Line 27). In this way, this clustering algorithm finds diverse samples from each cluster. The number of clusters and the nearest NSBRs to each cluster centroid guarantee the feature diversity and variety.

Algorithm 3 Find diverse NSBRs from each cluster.
SelRepreNSBR(K, S, C, l, n)

Input: The selected range K, the training set S, corpus C, the set length of each term l, the number n of SNBRs in each cluster
Output: The chosen NSBRs for training

```
1: $senVec \leftarrow$ **CalVec**($S$, $C$, $l$)
2: for $k$ in $K$ do
3:     $centroids_k \leftarrow ranSelect(k)$
4:     // Randomly chose the number of centroids $k$ in $S_{nsbr}$;
5:     $distMatrix[*,2]_k \leftarrow M_{init}$
6:     // each term includes the cluster's index and the distance from its centroid;
7:     $centroidChanged \leftarrow true$
8:     // set the initial state whether the centroids change;
9:     while $centroidChanged == true$ do
10:        $centroidChanged \leftarrow false$
11:        for $i \in S_{nsbr}$ do
12:            // Calculate the cluster index to which each NSBR $i$ should belongs,
13:            and the distance between the NSBR $i$ and the cluster;
14:            $Index_i \leftarrow calIndex(senVec_{nsbr}, centroids_k)$
15:            $Dist_i \leftarrow calDist(senVec_{nsbr}, centroids_k)$
16:            if $distMatrix_k(i,0) != Index_i$ then
17:                $centroidChanged \leftarrow true$
18:            end if
19:            $distMatrix_k(i,0) \leftarrow Index_i$
20:            $distMatrix_k(i,1) \leftarrow Dist_i$
21:        end for
22:        if $centroidChanged == true$ then
23:            $centroids_k \leftarrow updateCtd(disMatrix_k)$
24:            // Update the centroids in each cluster;
25:        end if
26:    end while
27:    $fltNSBR_k \leftarrow topN(centroids_k, disMatrix_k, n_k)$
28: end for
29: return $fltNSBR_k$
30: // Return the retained NSBRs for different $k$ and $n$.
```

3.3. Model training and prediction

Model Training. The CNN–BLSTM mechanism can excellently capture long-term dependencies, and the Attention mechanism can compensate for the former's neglect of different levels of attention to terms. In the clustering stage, each SBR is expressed as a tf–idf-weighted word embedding vector for clustering and filtering. However, in the model training stage, to capture the sequential and semantic information, the relationship between terms in bug reports (SBR and NSBR) should be explored as we mentioned in Section 2.4.2. Thus, for model training, we no longer regard a report as a whole but consider each term in each retained text as a unit for further analysis. These terms are represented as word embeddings via $word2vec$ technique (as same as that is used in the clustering stage) to form our embedding layer, which requires that the vectors of terms should have the same length for further convolution calculation.

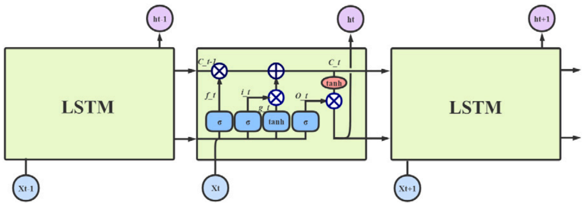


Fig. 3. The structure of the LSTM cell.

Due to the small size of our preprocessed datasets, in the front part of the whole model, we choose the shallow CNN-1D architecture to capture the contextual information to avoid high complexity or insufficient training of model parameters. Each term in the training set is calculated and padded into a d-dimensional sequence in Algorithm 1, and the variable-length sequence composed of d-dimensional vectors ($None$, $vector\_dim$) is sent to the *embedding layer* as the input. The trained weights are also used to generate a 3D tensor ($None$, $vector\_dim$, $embedding\_size$), which is incorporated into the *convolution layer* with fixed-size kernels to capture features from temporal contextual information. This layer outputs a tensor with ($None$, new_steps, $filters$) shape. Followed by a *max pooling layer* with output shape ($None$, $downsampled\_steps$, $filters$), the role of this pooling layer is to filter the features extracted by the model, which can not only remove some redundant information but also reduce the number of nodes in the network, thereby reducing the number of training parameters.

Afterward, a *Bidirectional LSTM layer* is added, taking the feature sequence calculated by the CNN architecture as input. In NLP, a significant feature of the text is serialization, and the order in which terms appear is often closely related to the semantics of the sentence. To make full use of the information of the sentence structure, we adopt a BLSTM model to extract the semantic information better, which applies memory cells to store the important information of long-range context. An LSTM unit illustrated in Fig. 3 consists of an input gate i_t, a forget gate f_t, an output gate o_t, and a memory cell c_t that acts as an accumulator of the state information. This memory cell can be accessed, written and cleared by those controlling gates. When a new input comes, if i_t is activated, the information will be accumulated to the cell. Similarly, if the f_t is turned on, the information of the last state c_{t-1} should be forgotten. The o_t controls whether the latest memory cell c_t will be propagated to the final state h_t. σ represents the logistic sigmoid activation function and W indicates the corresponding weight matrix from the cell to gate vectors. At each specific time node, the network can choose to remember or forget some information and send it to the next moment, and the steps are demonstrated by Eqs. (2)–(7) [73,82].

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{2}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{3}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{4}$$

$$g_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{5}$$

$$c_t = f_t c_{t-1} + i_t g_t \tag{6}$$

$$h_t = o_t \tanh(c_t) \tag{7}$$

Since the network contains two sub-networks, which are forward and backward pass respectively, the output of the ith term is represented in Eq. (8), and the shape of the output is ($None$, $timesteps$, $units \times 2$).

$$h_i = \overrightarrow{h_l} \oplus \overleftarrow{h_l} \tag{8}$$

Table 1
Details of the collected datasets.

Project	Time period	BRs	SBRs	SBRs(%)
Chromium	08/30/2008-06/11/2010	41,940	192	0.5
Camel	07/08/2007-09/18/2013	1000	32	3.0
Ambari	09/26/2011-08/08/2014	1000	29	3.0
Derby	09/28/2004-09/17/2014	1000	88	9.0
Wicket	10/20/2006-11/09/2014	1000	10	1.0

Because we adopt the self-attention model, the attention mechanism is able to dynamically generate the weights of different connections, which can handle variable-length information sequences. In our model, the input of our *Attention layer* can be expressed by three same fixed-length vectors query (Q), key (K), and value (V). By calculating the similarity score between two matrices (Q and K), applying *softmax* function to activate it, and getting the weighted score of each input vector through dot product, we can finally gain the result keeping the same shape. This process is actually a manifestation of the Attention mechanism that eases the complexity of the prediction model. It is not necessary to input all input information into the next layer for calculation, only need to select some task-related information.

$$Attention(Q, K, V) = softmax(Q \cdot K^T) \cdot V \tag{9}$$

Prediction. In the end, the output of this stage is sequentially put into the *flatten* and *dense* layer with $sigmoid$ function. The output is the predicted labels of the training data. Besides, the prediction model with the trained parameters is also saved for the next step. Through word embedding and padding, each term in the test data is represented by a fixed-length vector. These vectors are sent to the trained model and evaluated. The overall training and prediction processes can be found in Fig. 2.

4. Empirical evaluation

We implemented our experiment on Windows 10, running on an Intel® Core™ i7-9700 CPU @3.00 GHz with 8 cores, and an NVIDIA GeForce RTX 2080-Ti GPU with VRAM size of 12 GB.

4.1. Research questions

We employ three state-of-the-art methods FARSEC, SMOTUEND, and LTRWES as our baselines. All of them applied the commonly used machine learning algorithms (listed in Section 2.4.1) to build models.

Our experiments are performed by addressing the following research questions (RQs):

- **RQ1**: How effective is CASMS in classifying bug reports compared with the three baselines?
- **RQ2**: How does CASMS find the best results effectively and how robust is it based on the number of clusters and the total number of selected NSBRs?
- **RQ3**: Does selecting diverse samples through clustering improve the results?

For the RQ1, we calculate the average of top five best results for each project since these baselines extract the best results from multiple combinations between filters and classifiers. To make the comparison fair enough, we list the average of the top five best results of FARSEC, SMOTUEND, LTRWES, and CASMS. To answer the RQ2, we draw figures to illustrate (1) how the Elbow method finds the optimal number of clusters k^* and (2) the laws of the ratio of the two-class data corresponding to the best g-measure. To verify the effectiveness of retaining diverse data (answer the RQ3), we implement the idea of choosing NSBRs that are the most dissimilar to SBRs (held by our baselines) on the same k-means clustering algorithm.

4.2. Datasets

In the experiments, we use the same five datasets as the three baselines [3,5,6] for a fair comparison. The datasets were collected by Ohira et al. [83] with a Comma Separated Value (CSV) format, where each row represents a bug report, and the columns describe the features of these reports including the metadata information and textual description (e.g., *issue id*, *priority*, *security*, *description*, and *summary*). Bug reports are classified according to the label of the security field, where 1 is for SBR and 0 is for NSBR. The details of datasets are shown in Table 1. We aggregate the information of description and summary for training, and an example is described in Fig. 1. The text in the orange font is *summary*, and the text in gray font is *description*. To calculate the $tf - idf$ vectors, for each project, the vocabulary is its training set. It aims to find the importance of words in the specific dataset. The large domain corpus used for implementing the word embedding technique in Algorithm 1 has been extracted from 1,490,066 bug reports and open-sourced in [6]. The five datasets and the corpus can be found in our GitHub.[1]

4.3. Experiment setup

Our prediction model is implemented by using TensorFlow [84], an open-sourced deep learning library using Python. In the process of designing our model, parameter setting has a significant effect on its performance. We use binary *Cross-Entropy* for labeling the bug reports into two categories. In addition, different *optimizers* for iteratively updating the network weights and minimizing the loss are applied to enhance the performance, and *regularizers* are implemented to alleviate overfitting problems. To make the model more adaptive and general, the *model check point* technique is used to keep tracking the performance of the model on the validation dataset at each epoch.

The bug reports are equally divided into training/validation and test parts. For each project, the 50% of the whole dataset is used for model training, and the remaining 50% is for model testing (prediction). During the model training stage, the K-fold cross-validation methodology is applied. Because our datasets are small, we choose K=3 for our experiments. The training/validation set is divided into three parts, one part is kept as the data for validation, and the other two parts are used for training. The cross-validation is repeated three times, each validation is performed once, the results of three times are averaged, and a single estimate is finally obtained. The model with the highest average g-measure on the validation dataset is saved for prediction.

For hyperparameter tuning, to reduce the high computational cost of grid search by generating possible combinations among different hyperparameters, we first refer to the machine learning tool scikit-learn [85] in Python to estimate the configurations that are expected to contribute to better results. Then, we determine the search scope (shown in Table 2) and tune the hyperparameters successively.

4.4. Evaluation metrics

We employ the same three evaluation metrics as the three baselines [3,5,6].

Table 3 lists the definitions of the metrics to evaluate the performance of prediction models, where TP (True Positive) is the number of SBRs that are correctly predicted as SBRs, FP (False Positive) is the number of NSBRs that are wrongly predicted as SBRs, FN (False Negative) is the number of SBRs that are wrongly predicted as NSBRs, and TN (True Negative) is the number of NSBRs that are correctly predicted as NSBRs. The probability of detection (pd) is the ratio of the correctly predicted SBRs to the actual SBRs; The probability of false alarm (pf) is the ratio of the NSBRs that are mislabeled as SBRs to all

[1] https://github.com/mkkmaomao/bug-report-prediction

Table 2
Hyperparameter tuning range for the prediction model.

Explanation	Architecture	Hyperparameter	Search scope
Embedding	embedding_size	[50, 100, 200]	The indexes representing terms are turned into vectors with the same length.
Convolution	filters	[64, 128]	The number of output filters in the convolution layer.
	kernel_size	[3, 5, 8]	A convolution kernel is used for producing a tensor of outputs, its size determines the length of convolution window.
	activation	[‘tanh’, ‘relu’, ‘sigmoid’, ‘leak relu’]	The activation function for the hidden layer.
B-LSTM	units	[32, 64, 128]	The dimension of the output of one LSTM layer (the BLSTM layer should have the units ×2-dimension).
	dropout	[0.1, 0.2, 0.5]	The fraction of the dropped units aiming to enhance the generalization ability.
	regularizer	kernel: [‘l_1’, ‘l_2’], bias: [‘l_2’]	The regularizer functions applied to avoid overfitting.
Model Compiling	optimizer	[‘adam’, ‘rmsprop’, ‘adadelta’, ‘nadam’]	The optimizers invoked for training.
	batch_size	[32, 64, 128]	The number of samples selected for one training, which affects the optimization degree and speed of the model.

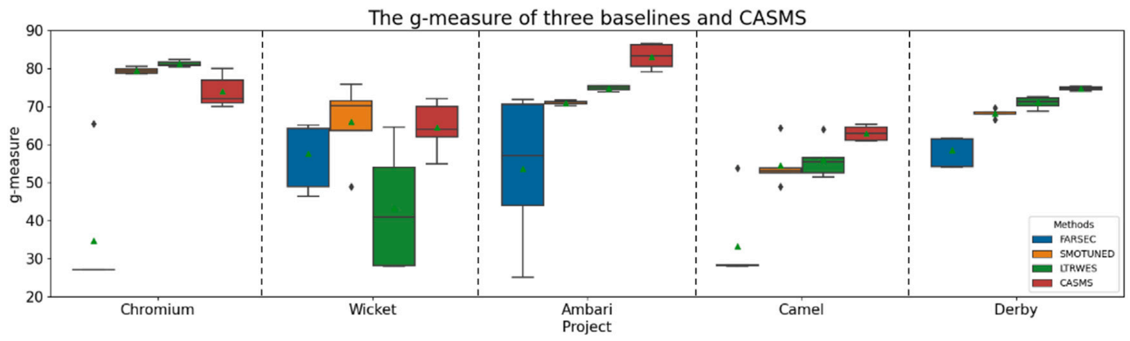


Fig. 4. The Boxplot of the g-measure values with CASMS and the three compared methods.

Table 3
The evaluation metrics used in our study.

	Prediction	Actual	
		SBRs	NSBRs
	SBRs	TP	FP
	NSBRs	FN	TN
pd	TP/(TP+FN)		
pf	FP/(FP+TN)		
g-measure	(2×pd×(100-pf))/(pd+(100-pf))		

Table 4
Comparison of the average g-measure, recall, and false alarm of FARSEC, SMOTUNED, LTRWES, and CASMS.

Project	Metric	FARSEC	SMOTUNED	LTRWES	CASMS
Chromium	g-measure	34.79	79.38	**81.20**	74.00
	pd	22.48	**78.42**	77.40	73.05
	pf	0.96	18.84	14.59	24.41
Wicket	g-measure	57.69	**66.02**	43.18	64.60
	pd	50.00	63.36	36.74	**73.33**
	pf	23.49	26.98	25.29	40.85
Ambari	g-measure	53.73	70.92	74.79	**83.12**
	pd	40.00	57.1	66.70	**82.85**
	pf	5.93	6.41	14.87	16.10
Camel	g-measure	33.34	54.53	56.00	**62.99**
	pd	23.36	52.22	53.32	**64.45**
	pf	15.47	39.74	38.09	37.05
Derby	g-measure	58.59	68.15	71.02	**74.72**
	pd	46.68	60.94	57.10	**70.00**
	pf	17.07	21.74	5.93	19.44
Overall	g-measure	47.63	67.80	65.24	**71.89**
	pd	36.50	62.41	58.25	**72.74**
	pf	12.58	22.74	19.75	27.57

NSBRs; The g-measure [86] is the harmonic mean of pd and (100-pf), where 100-pf represents the specificity (i.e., not predicting NSBRs as SBRs). The g-measure is a comprehensive metric, and more suitable for evaluating the prediction models trained on the unbalanced data [6]. Therefore, the g-measure is also considered as the optimization target for training our prediction model. To evaluate the robustness of our approach CASMS, we introduce the standard deviation (SD) indicator. In statistics, the standard deviation is a measure of the amount of variation or dispersion of a set of values [87]. A low standard deviation indicates that the values tend to be close to the mean (also called the expected value) of the set, while a high standard deviation indicates that the values are spread out over a wider range. The calculation of SD is shown below.

$$SD = \sqrt{\frac{1}{N}\sum_{i=1}^{n}(x_i - \mu)^2}, \tag{10}$$

where N is the number of experiments, x_i represents the value of g-measure in the ith experiment, and μ indicates the corresponding average value of all experiments.

5. Experimental results

5.1. RQ1: How effective is CASMS in classifying bug reports compared with the three baselines?

Methods: There are the six filters and five classifiers used in FARSEC and SMOTUNED, which generates the 30 (=6 × 5) combination results. In each combination, there are different resampling ratios of NSBRs to SBRs based on their set threshold for each project. Similarly, LTRWES applies two kinds of selectors and six classifiers. For each

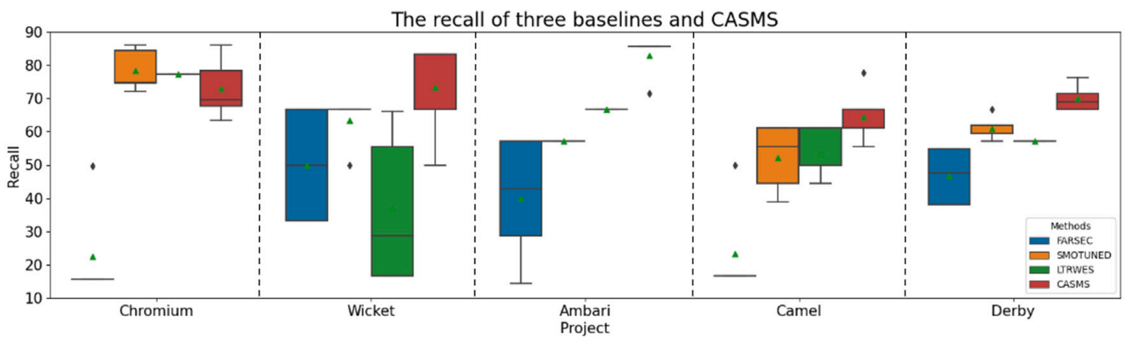


Fig. 5. The Boxplot of the recall values with CASMS and the three compared methods.

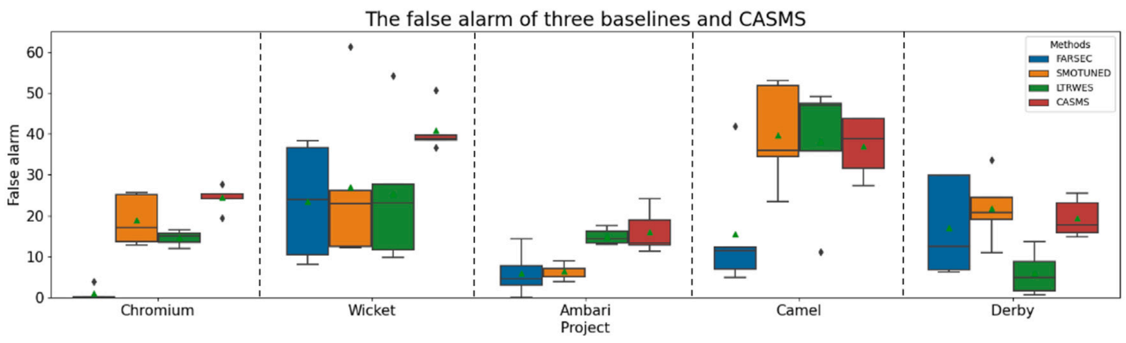


Fig. 6. The Boxplot of the false alarm values with CASMS and the three compared methods.

project, in each of the 12 (=2 × 6) combinations, the resampling ratio of NSBRs to SBRs is from 1:1 to 10:1. For these three approaches and CASMS, we illustrate the performance of the top five best results via Boxplots shown in Fig. 4, 5, and 6, and we also detail the average of three metrics of the top five best results in Table 4.

Results: CASMS achieves the highest average g-measure value (71.89%) on the five projects, and outperforms FARSEC, SMOTUNED, and LTRWES by 24.26%, 4.09%, and 6.65%, respectively.

As shown in Fig. 4, SMOTUNED and LTRWES perform better than CASMS in g-measure, and far surpass FARSEC on the *Chromium* project. Compared with FARSEC and LTRWES, SMOTUNED and CASMS perform well on the *Wicket* project with a smaller variance. On the remaining projects *Ambari*, *Camel*, and *Derby*, CASMS has outstanding performance, which outperforms three baselines in both the mean and median values of g-measure. Since the primary purpose of SBRs detection is to predict the actual SBRs more correctly, SBRs usually have the priority to be detected than NSBRs. Therefore, the pd is a more useful metric than pf, as it describes the ability of the prediction model to identify the actual SBRs from all bug reports. Fig. 5 displays the recall values (pd) of three methods and CASMS. Except for the *Chromium* project where SMOTUNED performs slightly better than CASMS, on the remaining four projects, CASMS significantly outperforms other methods with an ideal small variance. Similarly, Fig. 6 illustrates the performance of three baselines and our approach in false alarm values (pf). On the *Chromium* and *Wicket* projects, CASMS has a relatively high average pf compared with other methods, but on the *Ambari*, *Camel*, and *Derby* projects, CASMS has a similar or even lower pf. It implies that CASMS achieves good performance on the g-measure and pd evaluation metrics at the expense of the performance of the pf metric to some extent, and the relatively high pf is still an issue in identifying security bug reports, which needs to be improved in our future work.

Intuitively, Table 4 lists the average g-measure, pd, and pf values of the five best combination results of FARSEC, SMOTUNED, LTRWES, and CASMS. Although the compared methods outperform CASMS (27.57%) by 4.83%–14.99% in terms of pf, CASMS is able to identify SBRs more accurately with high pd (72.74% on average), which outperforms FARSEC, SMOTUNED, and LTRWES by 36.24%, 10.33%, and 14.49%, respectively. In terms of g-measure, CASMS performs better than the three baselines on the three projects *Ambari*, *Camel*, and *Derby* by 8.33%–29.39%, 6.99%–29.65%, and 4.09%–16.13%, respectively. On the *Wicket* project, the g-measure value of CASMS is 6.91% and 21.42% higher than that of FARSEC and LTRWES, separately, and only 1.42% lower than that of SMOTUNED. On the *Chromium* project, the performance of CASMS is 39.21% better than FARSEC, but worse than SMOTUNED and LTRWES by 5.38% and 7.2%, respectively.

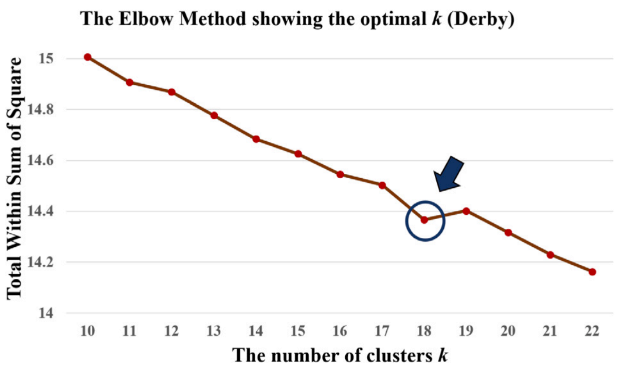


Fig. 7. Selection of an optimal k^* in the *Derby* project.

Answer to RQ1: CASMS is superior to FARSEC, SMOTUNED, and LTRWES in average g-measure and recall.

Table 5
Comparison of the recommended k^* values and the experimental k values of each project.

Project	Recommended k^*	Calculated k with best performance
Chromium	**18**, 20	15, **18**
Wicket	**4**	**4**, 5
Ambari	**4, 8**	**4, 8**
Camel	**4**	3, **4**
Derby	**18, 19**	**18, 19**

Table 6
The ratio of two-class data used for training corresponding to the best g-measure of each project.

Project	NSBRs	SBRs	Ratio
Chromium	180	77	**2:1-3:1**
	150	77	**1:1-2:1**
Wicket	10	4	**2:1-3:1**
	12	4	**3:1**
Ambari	36	22	**1:1-2:1**
	38	22	**1:1-2:1**
Camel	21	14	**1:1-2:1**
	24	14	**1:1-2:1**
Derby	52	47	**1:1-2:1**
	47	47	**1:1**

Table 7
Comparison of the average (%), standard deviation (%), and median (%) of g-measure of LTRWES and CASMS.

Project	Indicator	LTRWES		CASMS
		rs-filter	ms-filter	
Chromium	avg	**79.23**	53.20	71.22
	sd	**2.17**	16.95	4.52
	md	**79.17**	58.83	70.00
Wicket	avg	13.25	32.20	**62.83**
	sd	11.74	15.49	**7.44**
	md	8.35	27.91	**62.42**
Ambari	avg	69.28	67.05	**77.69**
	sd	**3.50**	9.56	5.39
	md	69.60	69.38	**77.37**
Camel	avg	47.57	44.69	**60.51**
	sd	7.19	11.49	**3.31**
	md	47.00	48.25	**60.70**
Derby	avg	61.20	66.16	**72.99**
	sd	5.78	6.58	**2.41**
	md	61.00	67.40	**73.36**
Overall	avg	54.11	52.66	**69.05**
	sd	6.08	12.01	**4.61**
	md	53.02	54.35	**68.77**

5.2. RQ2: How does CASMS find the best results effectively and how robust is it based on the number of clusters and the total number of selected NSBRs?

Methods (1): To avoid repeated experiments, CASMS roughly estimates the optimal number of clusters k^* using the Elbow algorithm (shown in Algorithm 2). To show the validity of the coarse estimate, the recommended k^* values are calculated and listed in the 2nd column of Table 5. The 3rd column list the k values when the g-measure reaches the optimal value, which is obtained through multiple experiments. Fig. 7 takes *Derby* dataset as an example, showing one of the recommended k^* that is equal to 18 (is circled and marked with a dark blue arrow).

Results (1): In Table 5, the k^* calculated ten times is 4 or 8 for the datasets with a few SBRs (*Camel*, *Ambari*, *Wicket*), and for other datasets with more SBRs (*Chromium* and *Derby*), the range of k^* is between 18 and 20.

As shown in the 3rd column of Table 5, it can be seen that the model achieves the best performance within a certain range of k value for each dataset. The three projects (*Camel*, *Ambari*, *Wicket*) containing less SBRs for training have k values ranging from 3 to 8, and the other two projects (*Derby* and *Chromium*) with more SBRs have k values ranging from 15 to 19. We can find that when the prediction results for each project are optimal, the values of k almost completely cover the value of k^* we estimated in advance (marked in bold). For example, in the best cases, the k values of *Ambari* (4 and 8) and *Derby* (18 and 19) totally coincide with the pre-calculated k^*.

Methods (2): To explore whether there is a certain rule of choosing the resampling ratio of two classes and whether it is robust, we calculate the best g-measure of different ratios and the corresponding indicator for evaluating the stability of CASMS. Table 6 shows the number of NSBRs and SBRs in the training set when the two best results reach, and the ratio of these two classes. Fig. 8 displays how does the g-measure change as the percentage of retained training NSBRs increases for each project while the range of k is from 2 to 20. Furthermore, we compare the three indicators (average, standard deviation, and median) between LTRWES and CASMS, and the standard deviation of g-measure is used to evaluate the robustness of these two approaches.

Results (2): The relationship between the percentage of retained NSBRs and the g-measure metric is illustrated in Fig. 8. It is evident that the four lines of *Wicket*, *Camel*, *Ambari*, and *Derby* have a general trend that it increases rapidly before reaching the highest g-measure value, followed by a monotonic decline. Table 6 lists the number of selected NSBRs where the peak values of g-measure reach, and the corresponding percentage of retained NSBRs are 2.02%, 4.32%, 7.53%, and 11.48%, respectively. Since the remaining project *Chromium* has more SBRs for training, this may make the choice of the number of NSBRs fall into a local optimal solution. There is not much difference between the local optimal g-measure and the globally optimal g-measure, with a difference of 3.85%. The percentage corresponding to the optimal number of NSBRs is 0.86%.

Overall, in all the optimal cases, the ratio of NSBRs to SBRs for training is between 1:1 and 3:1. *Chromium* has the ratio range from 1:1 to 3:1, *Wicket* has the ratio range from 2:1 to 3:1, and the others have the ratio range from 1:1 to 2:1.

Method (3): In order to evaluate the robustness of CASMS, for each project, we calculate the standard deviation of g-measure in LTRWES and CASMS. The average and median indicators are also introduced for comparison. Since LTRWES conducted experiments under the ratio of two classes from 1:1 to 10:1, we calculate the g-measure under the ratio span and choose the best-performing SVM as the classifier in LTRWES. For CASMS, we use the results under the ratio span between 1:1 and 3:1 for comparison. In Table 7, to avoid too many decimals, the "%" of the three indicators (average, standard deviation, and median) shown here is omitted.

Result (3): The best results are marked in bold, and we have two main findings. The first finding is that the maximum standard deviation of CASMS is 7.44%, and that of LTRWES is 16.95%. For the three projects (*Wicket*, *Camel*, and *Derby*), the standard deviation of CASMS is much lower than that of LTRWES. For the other projects (*Chromium* and *Ambari*), when LTRWES uses rs-filter, the standard deviation of CASMS is 2.1 and 1.5 times that of LTRWES, respectively; When ms-filter is used, the standard deviation of LTRWES is 3.8 and 1.8 times that of CASMS, respectively.

The second finding is that the average and the median of CASMS on the four projects (except *Chromium*) are higher than LTRWES, and the gap between CASMS and LTRWES is large. On the *Wicket* project, the median and the average g-measure of CASMS exceed those of LTRWES by 49.58% and 54.07%, respectively. On the *Chromium* project that CASMS does not perform as well as LTRWES, but the difference of performance between CASMS and LTRWES is small, i.e., the median and average of CASMS are 8.01% and 9.17% lower than those of LTRWES.

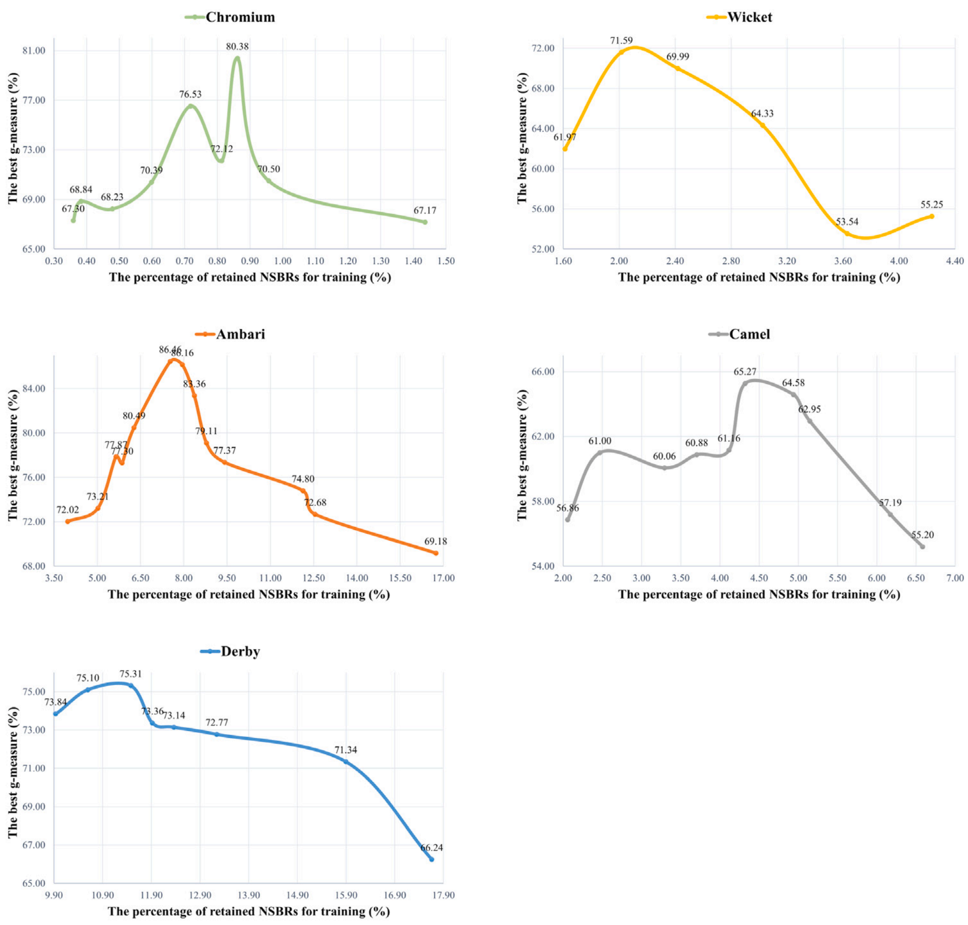


Fig. 8. Changes in the best g-measure with the number of NSBRs retained.

On the whole, the standard deviation of g-measure of CASMS is 1.47% and 7.4% lower than that of LTRWES with rs-filter and ms-filter, respectively. The average and median of g-measure of CASMS are higher than those of LTRWES by 14.94%–16.39% and 14.42%–16.75%.

Answer to RQ2: CASMS can narrow the range of cluster numbers based on the Elbow method and quickly find the optimal number of NSBRs (should be one to three times the number of SBRs for projects), which indicates the effectiveness. In addition, the robustness of CASMS is better than that of LTRWES.

5.3. RQ3: Does selecting diverse samples through clustering improve the results?

Methods: To be more rigorous, we also test the idea of choosing the NSBRs with the most dissimilar to SBRs held by our baselines through k-means clustering algorithm. Both NSBRs and SBRs are used for clustering. When the value range of k is 2 to 20, there are 19 groups of NSBRs. For each value of k, we take the union of the NSBRs in the selected clusters that contain fewer SBRs but more NSBRs. For example, if the ratio of NSBRs to SBRs in a cluster is greater than 10, the NSBRs in the cluster will be gathered. In other words, if there are many NSBRs but a few SBRs in a cluster, these NSBRs in this cluster will be regarded as the most dissimilar to SBRs. In order to keep the ratio of NSBRs to SBRs between 1:1–3:1, we then take the intersection of the NSBRs obtained when k is a different value. After that, the intersecting NSBRs in these clusters are integrated with all SBRs for training.

In detail, after each clustering with different k, two dictionaries (pairs of *key* and *value*) are generated for NSBRs and SBRs. In each dictionary, the pairs are arranged in reverse order according to the number of samples in each cluster. *key* refers to the cluster index assigned to the samples, and the *value* is the ranking of the number of samples corresponding to the cluster. The algorithm then traverses the same cluster index in each pair of two dictionaries to see if the cluster meets a certain condition. For example, if the difference between the ranking of the same cluster in the dictionary of SBRs and in that of NSBRs is greater than a certain value, the cluster indexes will be stored. Finally, the NSBRs in the selected clusters with different k value are intersected and the final filtered NSBR data is obtained. If the number of screened samples is insufficient, more samples can be filtered out by adjusting the threshold. Finally, for each dataset, the number of selected NSBRs that are the most dissimilar to SBRs and the number of diverse NSBRs are kept in the same range, e.g., the ratio of selected NSBRs to SBRs is between 1:1 and 3:1.

In Fig. 9, the best g-measure values based on two types of filtering methods are illustrated by a bar graph. The orange bars represent the results of retaining the diverse NSBRs, and the blue bars indicate the results of retaining the NSBRs that are the furthest to SBRs.

Results: When adopting the prediction model with the same parameters (Section 3.2), the best results in terms of pd, pf and g-measure for each project are not as good as the method of retaining diverse NSBRs. In Fig. 9, the best pd, pf, and g-measure of the two types of date filters are compared, and the data filter of retaining the diverse NSBRs performs better in each of these metrics. Based on the data filter of retaining the dissimilar NSBRs, the overall g-measure and pd values are nearly 16.2% and 17.8% lower than those of CASMS, and the pf is nearly 12.2% higher than that of CASMS. This indicates that owing

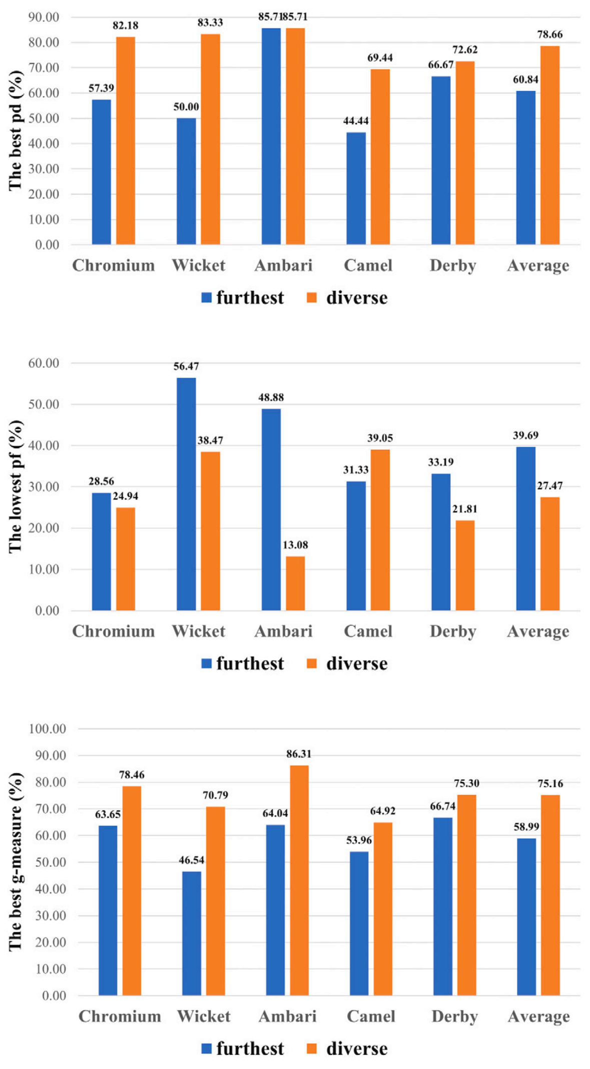


Fig. 9. The best pd, pf, and g-measure of each project based on two ideas of data selection.

to the high feature overlapping, even if the NSBRs in the clusters that are farther from most of the SBRs are retained, it is still challenging to achieve the desired effect we expect.

> Answer to RQ3: The performance of extracting the furthest NSBRs from SBRs is not a patch on that of selecting diverse NSBRs.

6. Threats to validity

Construct Threats. One of the threats is the choice of evaluation metrics, which is very important for statistical analysis. Since detecting SBRs has the priority in bug report classification, the g-measure and recall are designed to be the widely used evaluation metrics. However, there are different number of combinations between filters and classifiers for our baselines (30 combinations of FARSEC and SMOTUNED, 12 combinations of LTRWES). If we only choose the best one result for comparison, it is not reasonable, e.g., compare a model that has spent a lot of effort on multiple parameter optimizations with a model with default parameters. To counter this threat, we calculate the average of the best five experiment results for comparison. And the Boxplots are drawn to comprehensively compare the performance between methods, which includes a number of indicators, such as the four quantiles and outliers on different metrics. In addition, data scarcity is another construct threat, such as lack of cybersecurity related description in the bug reports. In the four projects, the dataset is relatively small and even some of the samples have a short description related cybersecurity. To the best of our knowledge, there is no related work to effectively solve this threat. Therefore, it is considered to be included in our further work.

Internal Threats. The first threat comes from the uncertainty of clustering. The centroids of clusters may be changed due to the random initialization setting. To eliminate the possible bias, we execute the clustering algorithm five times for each project and extract different numbers of NSBRs. The selected samples for each time have a high repetitive rate of more than 80%, and the predicted results based on these samples have shown similar trends. Thus, the centroids formed by each clustering will not have a visible impact on the selected training data and the prediction results. The second one is that the performance of CASMS is sensitive to the selection of the cluster and NSBR numbers. To mitigate the threat, we conduct numerous experiments with a comprehensive combination of two quantities (Fig. 8) and calculate the recommended cluster numbers ten times for each project (Table 5) to ensure that the results shown in Table 4 are scientific and well-tried.

Conclusion Threats. Normally in experiments in similar fields, the ratio of training to test sets is set to 7:3 or 8:2 because abundant data is needed for training. In our datasets, there are limited samples (i.e., there are only a thousand samples in each of the four projects), especially there are a few SBRs in each project. Thus, not only do we need a certain number of SBRs for training, but we also need enough SBRs in the test set to evaluate the prediction results. We have tried creating more SBR samples to alleviate this threat via synthesizing new SBR samples or copying existed SBRs, but the results were not ideal and required continuously-tuning. Finally, we divide the ratio of training and test sets into 1:1 that is the same as our three baselines.

External Threats. The external threat relates to the generalizability of the findings to other datasets. The facts that (1) the dataset used in this study is constructed by five projects containing 45,940 bug reports and (2) the projects were collected over multiple years, allow for a degree of external validation. Despite the dataset reflects the class imbalance problem in bug reports, and similar conclusions are drawn across all five projects, the model generated may not be applicable in a more complex scenario. Hence, this work is showing the proposed technique is valid and outperforming other approaches under certain conditions, and its generalization for different datasets will be subject to scrutinization in the future study.

7. Conclusion

In this paper, we have proposed CASMS, a novel approach for detecting SBRs. The key challenges for classifying bug reports are that data with different labels have a high degree of feature overlap, and the contextual and sequential information cannot be captured. We designed a k-means clustering algorithm combined with the Elbow algorithm to collect the closest NSBRs to the cluster centroids as the retained diverse samples for training. The Elbow algorithm was used successfully to identify the optimal k for clustering. We also adopted a CNN–BLSTM attention model extracting the semantic and sequential information to enhance the prediction results. The experimental results have shown that CASMS outperforms the baselines by 4.09%–24.26% and 10.33%–36.24% in terms of the average g-measure and *recall*, respectively. In addition, CASMS only needs a small number of experiments to reach the results, and at the same time it has superior robustness.

Declaration of competing interest

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

Acknowledgments

This work is supported in part by the General Research Fund of the Research Grants Council of Hong Kong (No. 11208017) and the research funds of City University of Hong Kong (7005028 and 7005217), and the Research Fund by Intel (9220097), the Natural Science Foundation of Chongqing City (cstc2021jcyj-msxmX1115), the project supported by Sanya Science and Education Innovation Park of Wuhan University of Technology (2020KF0059), and the funding supports from other industry partners of City University of Hong Kong (9678149, 9440227, 9440180, 9220103, 9229029 and 9229098).

References

[1] M. Bozorgi, L.K. Saul, S. Savage, G.M. Voelker, Beyond heuristics: learning to classify vulnerabilities and predict exploits, in: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2010, pp. 105–114.
[2] M. Gegick, P. Rotella, T. Xie, Identifying security bug reports via text mining: An industrial case study, in: 2010 7th IEEE Working Conference on Mining Software Repositories, MSR 2010, IEEE, 2010, pp. 11–20.
[3] F. Peters, T.T. Tun, Y. Yu, B. Nuseibeh, Text filtering and ranking for security bug report prediction, IEEE Trans. Softw. Eng. 45 (6) (2017) 615–631.
[4] K. Goseva-Popstojanova, J. Tyo, Identification of security related bug reports via text mining using supervised and unsupervised classification, in: 2018 IEEE International Conference on Software Quality, Reliability and Security, QRS, IEEE, 2018, pp. 344–355.
[5] R. Shu, T. Xia, L. Williams, T. Menzies, Better security bug report classification via hyperparameter optimization, 2019, arXiv preprint arXiv:1905.06872.
[6] Y. Jiang, P. Lu, X. Su, T. Wang, LTRWES: A new framework for security bug report detection, Inf. Softw. Technol. 124 (2020) 106314.
[7] N. Jalbert, W. Weimer, Automated duplicate detection for bug tracking systems, in: 2008 IEEE International Conference on Dependable Systems and Networks with FTCS and DCC, DSN, IEEE, 2008, pp. 52–61.
[8] C. Sun, D. Lo, X. Wang, J. Jiang, S.-C. Khoo, A discriminative model approach for accurate duplicate bug report retrieval, in: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1, 2010, pp. 45–54.
[9] J. Deshmukh, K. Annervaz, S. Podder, S. Sengupta, N. Dubash, Towards accurate duplicate bug retrieval using deep learning techniques, in: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2017, pp. 115–124.
[10] N. Ebrahimi, A. Trabelsi, M.S. Islam, A. Hamou-Lhadj, K. Khanmohammadi, An HMM-based approach for automatic detection and classification of duplicate bug reports, Inf. Softw. Technol. 113 (2019) 98–109.
[11] B.S. Neysiani, S.M. Babamir, M. Aritsugi, Efficient feature extraction model for validation performance improvement of duplicate bug report detection in software bug triage systems, Inf. Softw. Technol. 126 (2020) 106344.
[12] Y. Tian, D. Lo, X. Xia, C. Sun, Automated prediction of bug report priority using multi-factor analysis, Empir. Softw. Eng. 20 (5) (2015) 1354–1383.
[13] T. Zhang, G. Yang, B. Lee, A. Chan, Predicting severity of bug report by mining bug repository with concept profile, in: Proceedings of the 30th Annual ACM Symposium on Applied Computing, 2015.
[14] Y. Tong, X. Zhang, Crowdsourced test report prioritization considering bug severity, Inf. Softw. Technol. (2021) 106668.
[15] X. Ye, R. Bunescu, C. Liu, Mapping bug reports to relevant files: A ranking model, a fine-grained benchmark, and feature evaluation, IEEE Trans. Softw. Eng. 42 (4) (2015) 379–402.
[16] A.N. Lam, A.T. Nguyen, H.A. Nguyen, T.N. Nguyen, Combining deep learning with information retrieval to localize buggy files for bug reports (n), in: 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE, 2015, pp. 476–481.
[17] Z. Li, Z. Jiang, X. Chen, K. Cao, Q. Gu, Laprob: A label propagation-based software bug localization method, Inf. Softw. Technol. 130 (2021) 106410.
[18] R. Almhana, M. Kessentini, W. Mkaouer, Method-level bug localization using hybrid multi-objective search, Inf. Softw. Technol. 131 (2021) 106474.
[19] M. Kim, E. Lee, ManQ: Many-objective optimization-based automatic query reduction for IR-based bug localization, Inf. Softw. Technol. 125 (2020) 106334.
[20] M. Hamill, K. Goseva-Popstojanova, Analyzing and predicting effort associated with finding and fixing software faults, Inf. Softw. Technol. 87 (2017) 1–18.
[21] K. Goseva-Popstojanova, J. Tyo, Experience report: security vulnerability profiles of mission critical software: empirical analysis of security related bug reports, in: 2017 IEEE 28th International Symposium on Software Reliability Engineering, ISSRE, IEEE, 2017, pp. 152–163.
[22] S. Panichella, G. Canfora, A. Di Sorbo, "Won't we fix this issue?" qualitative characterization and automated identification of wontfix issues on GitHub, Inf. Softw. Technol. (2021) 106665.
[23] K. Goseva-Popstojanova, A. Perhinschi, On the capability of static code analysis to detect security vulnerabilities, Inf. Softw. Technol. 68 (2015) 18–33.
[24] S. Kim, H. Zhang, R. Wu, L. Gong, Dealing with noise in defect prediction, in: 2011 33rd International Conference on Software Engineering, ICSE, IEEE, 2011, pp. 481–490.
[25] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, SMOTE: synthetic minority over-sampling technique, J. Artificial Intelligence Res. 16 (2002) 321–357.
[26] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (4) (1997) 341–359.
[27] W. Fu, T. Menzies, X. Shen, Tuning for software analytics: Is it really necessary? Inf. Softw. Technol. 76 (2016) 135–146.
[28] C. Sun, D. Lo, S.-C. Khoo, J. Jiang, Towards more accurate retrieval of duplicate bug reports, in: 2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, IEEE, 2011, pp. 253–262.
[29] J. Lilleberg, Y. Zhu, Y. Zhang, Support vector machines and word2vec for text classification with semantic features, in: 2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing, ICCI* CC, IEEE, 2015, pp. 136–140.
[30] J. Ramos, et al., Using tf-idf to determine word relevance in document queries, in: Proceedings of the First Instructional Conference on Machine Learning, vol. 242, (1) Citeseer, 2003, pp. 29–48.
[31] A. Aizawa, An information-theoretic perspective of tf–idf measures, Inf. Process. Manage. 39 (1) (2003) 45–65.
[32] D. Wijayasekara, M. Manic, J.L. Wright, M. McQueen, Mining bug databases for unidentified software vulnerabilities, in: 2012 5th International Conference on Human System Interactions, IEEE, 2012, pp. 89–96.
[33] M. Esposito, E. Damiano, A. Minutolo, G. De Pietro, H. Fujita, Hybrid query expansion using lexical resources and word embeddings for sentence retrieval in question answering, Inform. Sci. 514 (2020) 88–105.
[34] R.A. Stein, P.A. Jaques, J.F. Valiati, An analysis of hierarchical text classification using word embeddings, Inform. Sci. 471 (2019) 216–232.
[35] V. Di Carlo, F. Bianchi, M. Palmonari, Training temporal word embeddings with a compass, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, (01) 2019, pp. 6326–6334.
[36] N. Garg, L. Schiebinger, D. Jurafsky, J. Zou, Word embeddings quantify 100 years of gender and ethnic stereotypes, Proc. Natl. Acad. Sci. 115 (16) (2018) E3635–E3644.
[37] V. Tshitoyan, J. Dagdelen, L. Weston, A. Dunn, Z. Rong, O. Kononova, K.A. Persson, G. Ceder, A. Jain, Unsupervised word embeddings capture latent knowledge from materials science literature, Nature 571 (7763) (2019) 95–98.
[38] X. Dai, M. Bikdash, B. Meyer, From social media to public health surveillance: Word embedding based clustering method for twitter classification, in: SoutheastCon 2017, IEEE, 2017, pp. 1–7.
[39] L. Xiao, G. Wang, Y. Zuo, Research on patent text classification based on word2vec and LSTM, in: 2018 11th International Symposium on Computational Intelligence and Design, vol. 1, ISCID, IEEE, 2018, pp. 71–74.
[40] J. Gao, Y. He, X. Zhang, Y. Xia, Duplicate short text detection based on word2vec, in: 2017 8th IEEE International Conference on Software Engineering and Service Science, ICSESS, IEEE, 2017, pp. 33–37.
[41] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, 2013, arXiv preprint arXiv:1301.3781.
[42] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, 2013, arXiv preprint arXiv:1310.4546.
[43] X. Rong, Word2vec parameter learning explained, 2014, arXiv preprint arXiv:1411.2738.
[44] C. Boutsidis, P. Drineas, M.W. Mahoney, Unsupervised feature selection for the k-means clustering problem, in: Advances in Neural Information Processing Systems, 2009, pp. 153–161.
[45] S. Alelyani, J. Tang, H. Liu, Feature selection for clustering: A review, Data Clust. (2018) 29–60.
[46] S. Xiang, F. Nie, C. Zhang, Learning a mahalanobis distance metric for data clustering and classification, Pattern Recognit. 41 (12) (2008) 3600–3612.
[47] T. Li, S. Ma, M. Ogihara, Entropy-based criterion in categorical clustering, in: Proceedings of the Twenty-First International Conference on Machine Learning, 2004, p. 68.
[48] F. Murtagh, P. Contreras, Methods of hierarchical clustering, 2011, arXiv preprint arXiv:1105.0121.
[49] P.-N. Tan, M. Steinbach, V. Kumar, Introduction to Data Mining, Pearson Education India, 2016.
[50] G. Seif, The 5 clustering algorithms data scientists need to know, Towards Data Sci. (2018).
[51] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., A density-based algorithm for discovering clusters in large spatial databases with noise, in: Kdd, vol. 96, (34) 1996, pp. 226–231.
[52] K. Khan, S.U. Rehman, K. Aziz, S. Fong, S. Sarasvady, DBSCAN: Past, present and future, in: The Fifth International Conference on the Applications of Digital Information and Web Technologies, ICADIWT 2014, IEEE, 2014, pp. 232–238.
[53] X. Xu, M. Ester, H.-P. Kriegel, J. Sander, A distribution-based clustering algorithm for mining in large spatial databases, in: Proceedings 14th International Conference on Data Engineering, IEEE, 1998, pp. 324–331.

[54] M. Huang, F. Bian, A grid and density based fast spatial clustering algorithm, in: 2009 International Conference on Artificial Intelligence and Computational Intelligence, vol. 4, IEEE, 2009, pp. 260–263.
[55] J.P. Gleeson, S. Melnik, A. Hackett, How clustering affects the bond percolation threshold in complex networks, Phys. Rev. E 81 (6) (2010) 066114.
[56] A. Dharmarajan, T. Velmurugan, Applications of partition based clustering algorithms: A survey, in: 2013 IEEE International Conference on Computational Intelligence and Computing Research, IEEE, 2013, pp. 1–5.
[57] D. Xu, Y. Tian, A comprehensive survey of clustering algorithms, Ann. Data Sci. 2 (2) (2015) 165–193.
[58] F. Liu, Y. Deng, Determine the number of unknown targets in open world based on elbow method, IEEE Trans. Fuzzy Syst. (2020).
[59] T.M. Kodinariya, P.R. Makwana, Review on determining number of cluster in K-means clustering, Int. J. 1 (6) (2013) 90–95.
[60] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, A.Y. Wu, An efficient k-means clustering algorithm: Analysis and implementation, IEEE Trans. Pattern Anal. Mach. Intell. 24 (7) (2002) 881–892.
[61] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: A proposed framework and novel findings, IEEE Trans. Softw. Eng. 34 (4) (2008) 485–496.
[62] Z. Harry, The optimality of naive bayes, in: FLAIRS2004 Conference, 2004.
[63] W. Afzal, Using faults-slip-through metric as a predictor of fault-proneness, in: 2010 Asia Pacific Software Engineering Conference, IEEE, 2010, pp. 414–422.
[64] E.J. Weyuker, T.J. Ostrand, R.M. Bell, Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models, Empir. Softw. Eng. 13 (5) (2008) 539–559.
[65] J.G. Shanahan, N. Roma, Improving SVM text classification performance through threshold adjustment, in: European Conference on Machine Learning, Springer, 2003, pp. 361–372.
[66] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
[67] C.M. Bishop, et al., Neural Networks for Pattern Recognition, Oxford University Press, 1995.
[68] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Trans. Inform. Theory 13 (1) (1967) 21–27.
[69] C. Zhou, C. Sun, Z. Liu, F. Lau, A C-LSTM neural network for text classification, 2015, arXiv preprint arXiv:1511.08630.
[70] J.L. Elman, Finding structure in time, Cogn. Sci. 14 (2) (1990) 179–211.
[71] J. Gao, J.-Y. Nie, G. Wu, G. Cao, Dependence language model for information retrieval, in: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2004, pp. 170–177.
[72] A. Graves, S. Fernández, J. Schmidhuber, Bidirectional LSTM networks for improved phoneme classification and recognition, in: International Conference on Artificial Neural Networks, Springer, 2005, pp. 799–804.
[73] A. Graves, N. Jaitly, A.-r. Mohamed, Hybrid speech recognition with deep bidirectional LSTM, in: 2013 IEEE Workshop on Automatic Speech Recognition and Understanding, IEEE, 2013, pp. 273–278.
[74] N.S. Madiraju, S.M. Sadat, D. Fisher, H. Karimabadi, Deep temporal clustering: Fully unsupervised learning of time-domain features, 2018, arXiv preprint arXiv:1802.01059.
[75] Y. Wu, W. Li, Automatic audio chord recognition with MIDI-trained deep feature and BLSTM-CRF sequence decoding model, IEEE/ACM Trans. Audio Speech Lang. Proc. 27 (2) (2018) 355–366.
[76] B. Liu, S. Li, ProtDet-CCH: protein remote homology detection by combining long short-term memory and ranking methods, IEEE/ACM Trans. Comput. Biol. Bioinform. 16 (4) (2018) 1203–1210.
[77] D. Quang, X. Xie, DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences, Nucleic Acids Res. 44 (11) (2016) e107.
[78] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, 2014, arXiv preprint arXiv:1409.0473.
[79] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, Y. Bengio, Show, attend and tell: Neural image caption generation with visual attention, in: International Conference on Machine Learning, PMLR, 2015, pp. 2048–2057.
[80] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, Y. Bengio, Attention-based models for speech recognition, 2015, arXiv preprint arXiv:1506.07503.
[81] U. Khandelwal, H. He, P. Qi, D. Jurafsky, Sharp nearby, fuzzy far away: How neural language models use context, 2018, arXiv preprint arXiv:1805.04623.
[82] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, W.-c. Woo, Convolutional LSTM network: A machine learning approach for precipitation nowcasting, in: Advances in Neural Information Processing Systems, 2015, pp. 802–810.
[83] M. Ohira, Y. Kashiwa, Y. Yamatani, H. Yoshiyuki, Y. Maeda, N. Limsettho, K. Fujino, H. Hata, A. Ihara, K. Matsumoto, A dataset of high impact bugs: Manually-classified issue reports, in: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, IEEE, 2015, pp. 518–521.
[84] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: A system for large-scale machine learning, in: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), 2016, pp. 265–283.
[85] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[86] Y. Jiang, B. Cukic, Y. Ma, Techniques for evaluating fault prediction models, Empir. Softw. Eng. 13 (5) (2008) 561–595.
[87] J.M. Bland, D.G. Altman, Statistics notes: measurement error, Bmj 312 (7047) (1996) 1654.