Lee and Seo *Hum. Cent. Comput. Inf. Sci.* (2020) 10:26
https://doi.org/10.1186/s13673-020-00229-7

Human-centric Computing and Information Sciences

RESEARCH **Open Access**

Improving bug report triage performance using artificial intelligence based document generation model

Dong-Gun Lee and Yeong-Seok Seo*

*Correspondence:
ysseo@yu.ac.kr
Department of Computer Engineering, Yeungnam University, 280 Daehak-Ro, Gyeongsan, Gyeongbuk 38541, Republic of Korea

Abstract

Artificial intelligence is one of the key technologies for progression to the fourth industrial revolution. This technology also has a significant impact on software professionals who are continuously striving to achieve high-quality software development by fixing various types of software bugs. During the software development and maintenance stages, software bugs are the major factor that can affect the cost and time of software delivery. To efficiently fix a software bug, open bug repositories are used for identifying bug reports and for classifying and prioritizing the reports for assignment to the most appropriate software developers based on their level of interest and expertise. Owing to a lack of resources such as time and manpower, this bug report triage process is extremely important in software development. To improve the bug report triage performance, numerous studies have focused on a latent Dirichlet allocation (LDA) using the k-nearest neighbors or a support vector machine. Although the existing approaches have improved the accuracy of a bug triage, they often cause conflicts between the combined techniques and generate incorrect triage results. In this study, we propose a method for improving the bug report triage performance using multiple LDA-based topic sets by improving the LDA. The proposed method improves the existing topic sets of the LDA by building two adjunct topic sets. In our experiment, we collected bug reports from a popular bug tracking system, Bugzilla, as well as Android bug reports, to evaluate the proposed method and demonstrate the achievement of the following two goals: increase the bug report triage accuracy, and satisfy the compatibility with other state-of-the-art approaches.

Keywords: Bug report triage, Software defect prediction, Latent Dirichlet Allocation, Artificial intelligence, Machine learning, Software engineering

Introduction

Along with the fourth industrial revolution, artificial intelligence, big data, Internet of Things, and cloud computing are emerging as cutting-edge technologies globally. In particular, artificial intelligence has unlimited potential to further improve the quality of human life and can solve several difficult engineering problems [1–12]. Moreover, this technology provides basic ideas to derive successful solutions to numerous problems encountered in the software development field.

As the size and complexity of software evolve, software defects are becoming inevitable. A software defect is an error, flaw, mistake, or fault in a computer program or system, producing incorrect or unexpected results [13]. These defects inconvenience users by causing malfunctions, i.e., software defects continuously decrease the quality of the software until the defect is fixed. Thus, defects are a significant issue that must be resolved for software quality improvement. Various methods for effectively detecting, fixing, and patching bugs have been investigated by software developers [14, 15]. For software projects, the ratio of the cost of software maintenance to the total project cost exceeds 50% [16–19]. Corrective maintenance addressing software defects accounts for 20% of all maintenance activities [20–22], and an improved efficiency in fixing defect will have a direct effect on the reduction in software development and maintenance costs. These issues are considered to be significant for software development companies.

During a software development, bug reports are written to effectively manage and fix software bugs when they are detected during the software life cycles. Bug reports are documents that detail the occurrence of defects with a specific format between developers and reporters. In general, information regarding the reporter, environment, and other data, including the priority and severity used in triage, are recorded in bug reports. Developers make substantial efforts to fix bugs and improve the communication with a user or quality assurance (QA) team using bug reports.

When defects occur during software development and maintenance, a software development manager often follows the defect life cycle, as shown in Fig. 1. Figure 1 summarizes the stages of the defect life cycle. The straight lines indicate parts that are manually applied by developers. The dotted lines are parts that can be automated by a system. First, once defects are detected, bug reports are written with the initial state "NEW," and the manager analyzes whether a bug is valid and has been duplicated. If a bug is not detected by the manager, the report is sent to the developer and the state is changed to "OPEN." Second, as a result of the developer's activity, the bug is considered "CLOSED" or "REOPENED." As shown in Fig. 1, several stages must be completed before developers start to modify the code. An important stage involves classifying bug reports for the bug report manager. The classification is divided into two classes: (1) textual classification, and (2) triage. Textual classification is based on texts, such as the title or body text. Triage is a classification method not based on texts but based on the priority or severity of the defects. Through textual classification, the bug reports classified with similar reports are assigned to developers that make modules of the defects that have occurred. Textual classification is also used to identify duplicate bug reports, which account for 30% of all bug reports [24]. Because developers cannot address all bug reports, a triage is essential for attaining the best maintenance efficiency within a limited period of time.

If these defect classification processes are accurately and smoothly applied, no problems will occur; if not, the effect on the fixing of software defects and maintenance will decrease. An incorrect textual classification causes a bug report to be reassigned to other developers; the bug report cannot be fixed until the reassignment is complete. Thus, an incorrect textual classification decreases the maintenance efficiency. In addition, a mis-triage creates a more critical problem. Because unimportant problems are processed first, urgent defects can be delayed. In an incorrect textual classification, after a developer requests that a bug report be reassigned, the developer can

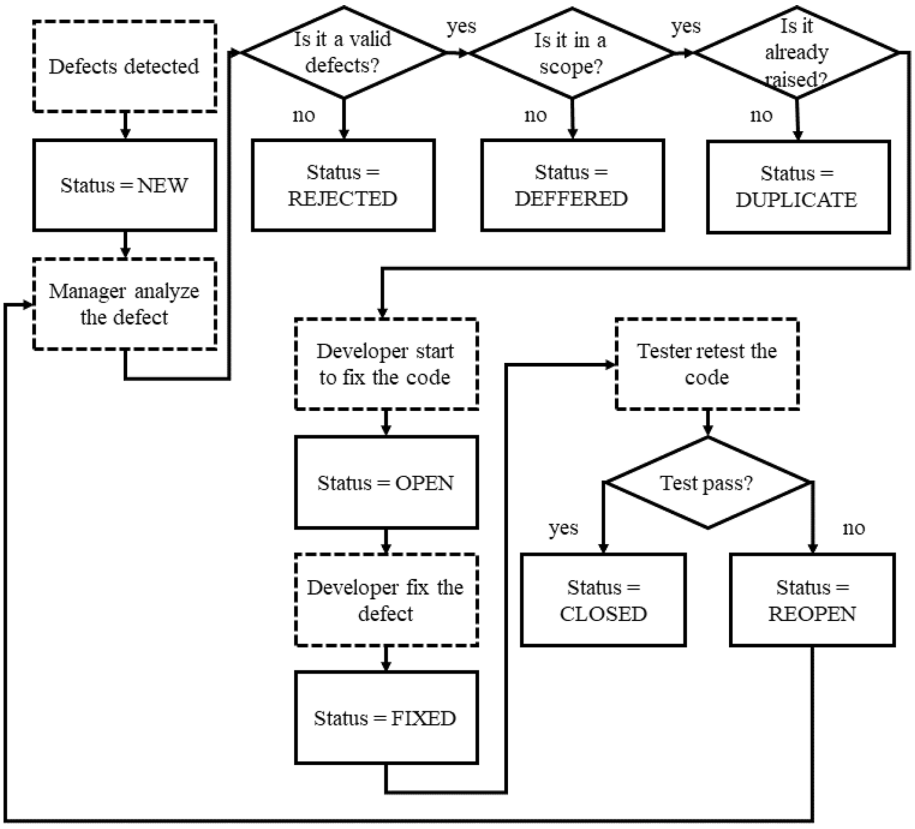


Fig. 1 Defect life cycle [23]

work on another job. Conversely, the cost of fixing unimportant defects caused by a mis-triage is irreversible. Thus, an accurate bug report classification and assignment are directly connected to software maintenance efficiency. Because this efficiency is connected to the cost incurred by the company, it is extremely important. Owing to the importance of an accurate bug report triage, a pre-existing bug report triage is manually applied. For example, in the case of Eclipse, developers may spend up to 2 h classifying bug reports every day.

To resolve these problems, artificial intelligence techniques are now actively being studied, and have shown better classification accuracy than traditional (non-artificial intelligence based) methods [25–36]. Such techniques can be a key to solving most of the current problems regarding this issue. Thus, there have been various attempts to overcome the weaknesses of traditional methods by combining artificial intelligence as a hybrid approach.

To reduce the effort required in this regard, studies have proposed the application of state-of-the-art automation methods for bug report classification [25–29]. In particular, latent Dirichlet allocation (LDA)-based classification methods are common because they are suitable to bug reports that contain text-based data. Although these methods are excellent in terms of textual classification, the accuracy of the triage is unsatisfactory. To achieve an improved LDA-based method, software engineers have proposed new methods that combine LDA with other approaches, such as the k-nearest neighbor (KNN) and a support vector machine (SVM) [30–35]. However, it is risky to combine LDA with other methods for improving the accuracy of bug report classification because the combined methods cannot be applied well when compatibility issues occur (e.g., a correlation or difference in the input data between the methods) between LDA and other approaches. The risk is greater when LDA is combined with another method. Thus, instead of combining LDA with another method to improve the performance of the bug report classification, the performance of LDA itself should be improved.

To improve the bug report triage performance, in this study, we focus on improving LDA itself and propose a new method based on multiple LDA and backpropagation techniques. The proposed method aims to improve the quality of the topic set produced through LDA classification. The method builds additional topic sets that complement the original topic set from a typical use of LDA, and classifies and analyzes them to support the original topic set for improving the accuracy of the bug report classification. To evaluate the proposed method, we use bug reports from Bugzilla [37] along with Android bug reports from Mining Software Repositories (MSR) [38, 39]. Any method that fails to classify a significant number of bug reports is useless, and we therefore verified that the proposed method is able to classify a significant number of bug reports as a repository platform. We also verified and determined the efficiency of the method for use in a bug triage. To determine the difference between the original LDA classification and the proposed method, we statistically verified the method using a paired T-test.

The main contributions of this study are as follows:

- A new method is proposed to improve the accuracy of bug report triage using multiple LDA and backpropagation techniques.
- The proposed method is able to maintain compatibility with the existing hybrid LDA methods through a design of the necessary conditions.
- Factors hindering the accuracy of the triage are identified through a detailed analysis.
- Our experiments were conducted based on bug reports for actual software used in practice.
- The superiority of the proposed method was validated through a statistical evaluation.

The remainder of this paper is organized as follows: Related studies are introduced in Sect. "Related work". Section "Background" describes the background information. Section "Approach" shows our method to improving the bug reports triage performance and avoid confliction with existing LDA-based triage methods. Section "Evaluation" evaluates the proposed method and Sect. "Discussion" discusses detailed analysis of the proposed method. This paper is concluded and future research is discussed in Sect. "Conclusions".

Related work

Bug report deduplication

Bug report deduplication is the process of removing duplicate bug reports. Duplicate bug reports cause an overestimation of the number of bug reports and increase the costs required. Thus, studies on bug report deduplication greatly help reduce the workload.

Alipour et al. [40, 41] used textual information (e.g., title, abstract or body text) to reduce bug report duplication. They proposed a BM25F based method that automatically extract the implications of the bug report and builds a dictionary (set of words). The researchers referred android layered architectural words [42], software non-functional requirements words [43], android topic words using LDA [44], android topic words using labeled-LDA [44] and random words in the English dictionary. As shown by the dictionary sources, the method is applied to android bug reports and an 11.55% performance improvement is achieved compared with REP [45]. A similar study [46] uses word embedding.

Aggarwal et al. [47, 48] improves the method in a study by Alipour [40] and proposes a method that is based on software engineering literature and reduces manual efforts for deduplication with minimal loss of triage accuracy. This study shows that the method of Aggarwal et al. is better than Alipour's method in Eclipse, Mozilla and Open Office.

Campbell et al. [49] focused on off-the-shelf information retrieval techniques. Although these techniques were not designed for bug reports, they outperformed other approaches in terms of crash bucketing (i.e., bug report grouping) at an industrial scale. The authors used more than 30 thousand report data from the Ubuntu repository and Mozilla's own automated system. Finally, they demonstrated that bug report deduplication still has significant room for improvement, particularly in terms of identifier tokenization through term frequency–inverse document frequency (TF–IDF).

Hindle et al. [50] proposed a method for preventing duplicate bug reports before they are submitted. This method finds duplicate or related bug reports in the bug database using texts. In addition, this simple method can be used to evaluate a new bug report deduplication method. This method is evaluated using bug reports from Android, Eclipse, Mozilla, and OpenOffice projects.

Nguyen et al. [51] proposed the DBTM, which has two advantages: both features are based on a topical method and information retrieval (IR). This method shows 20% performance improvement compared with the Relational Topic Model (RTM) [52] and REP [45] in Eclipse, Mozilla and Open Office.

Tian et al. [53] improve the study of Jarbert [54] and introduce three kinds of approaches. The first approach does not use term appearance (e.g., TF-IDF) but applies BM25 because BM25 is the best method according to the technical literature search. The second approach uses "product" as metadata, i.e., this method uses the notion that bug reports with different product are not duplicated. The third approach uses a comparison of the top k-similar bug reports instead of the most similar bug reports. This method improves the true positives and maintains low false negatives compared with a study of Mozilla projects by Jarbert.

Other machine learning methods [55, 56], such as hidden Markov models (HMMs) or deep networks, are proposed. They build a model that identifies the features of duplicate bug reports and utilize it. A multi-factor analysis method [55] that employs LDA, LNG and n-gram is also proposed.

Bug report triage

Bug report triage is a type of classification process. Because the developer's workload is limited, critical bug reports should be processed earlier. Thus, a bug report triage is a classification process based on "priority."

Tamrawi et al. [57] proposed Bugzie, which recommends bug reports. Bugzie builds fuzzy sets that are based on words extracted from the title and the description. Bugzie shows that it outperforms naïve Bayes, C4.5 (decision tree) and SVM with regarding to the temporal efficiency in Eclipse.

Wang et al. [58] proposed FixerCache, which is an unsupervised bug triage method. FixerCache overcomes the limits of supervised classification based on the activities of developers. FixerCache uses TF extracted from the title and the description of bug reports and outperforms naïve Bayes and SVM regarding the accuracy of classification.

Wen et al. [59] proposed Configuration Bug Learner Uncovers Approved options (CoLUA). CoLUA is a two-phase method that utilizes machine learning, IR and natural language processing (NLP) to resolve communication problems between developers and reporters. In the first phase, CoLUA determines what the bug report intends to convey based on its text information. In the second phase, CoLUA identifies the options that affect the communication in the labeled bug reports. The researchers evaluated CoLUA; their findings indicate that CoLUA has a better F-measure than the ZeroR classifier.

Zhang et al. [60] proposed the k-NN search and heterogeneous proximity (KSAP). KSAP employs the heterogeneous network of the bug report repository and historical bug reports to improve the auto-allocation of bug reports. KSAP is a two-phase method. First, KSAP obtains historically similar bug reports. Second, KSAP ranks the contribution of developers by heterogeneous proximity. The developers evaluated KSAP using Eclipse, Mozilla, Apache Ant and Apache Tomcat6. KSAP shows a performance improvement of 7.5–32.25% compared with ML-KNN [61, 62], DREX [63], DRETOM [64], Bugzie [57], and DevRec [62].

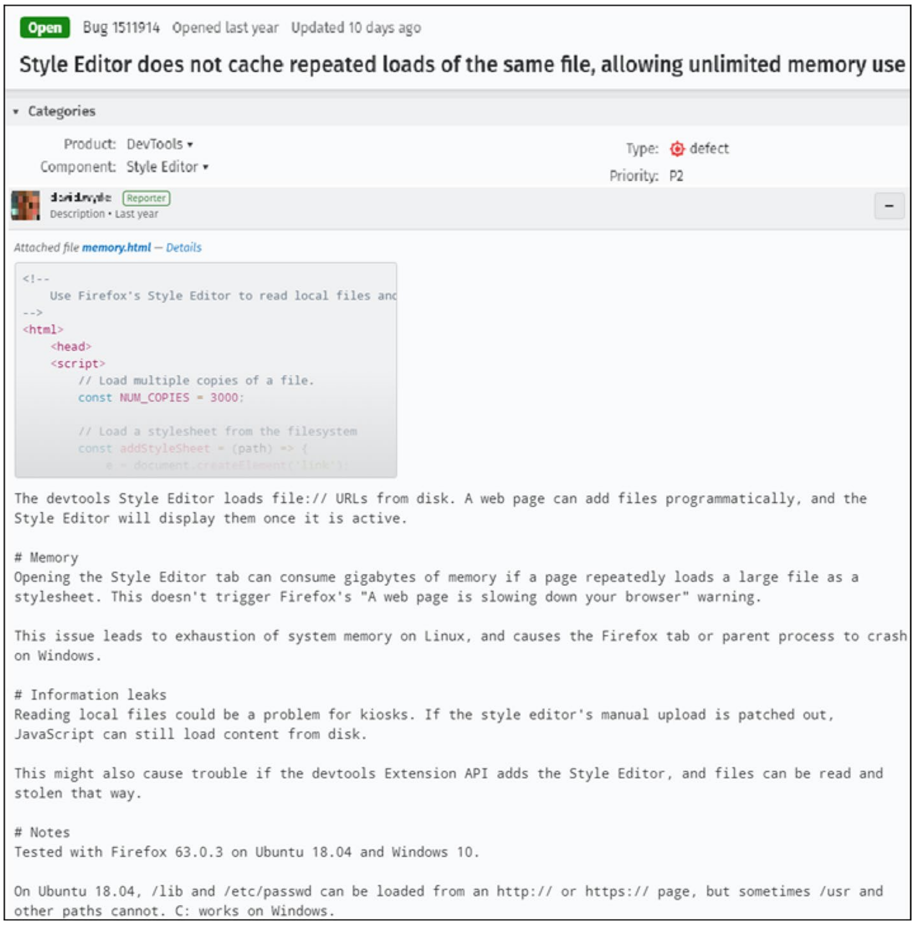
Many bug report triage methods [65–70] use data reduction. To achieve data reduction, these methods use KNN, naïve Bayes, and clustering and reduce feature selection and instance selection using the representative and statistic value of these methods or newly define "module selection."

Machine-learning based methods applied to bug triage have also been frequently studied. Florea et al. [71] proposed an SVM-based bug report assignment recommender implemented in a cloud platform that achieves better results than other SVM-based bug report assignment recommending systems. They evaluate their method using actual datasets consisting of Netbean, Eclipse, and Mozilla projects. Popular deep-learning-based methods, which machine-learning type approaches, have recently been proposed using two deep-learning classifiers, namely, convolutional and recurrent neural networks for a parallel and extendable recommending system [72], and using a convolutional neural network and word embedding for automated bug triage [73]. These studies use an actual open-source dataset and demonstrate a higher accuracy than existing machine-learning-based methods.

Background

Bug report

In a modern environment, bug reports are operated as a part of community of issue (bug) tracking systems, i.e., bug reports are identified by not only the developers or report managers in charge but also all related people and are even used as public data. Thus, the bug report process requires accurate classification to satisfy the needs of numerous people. A distinct difference is the metadata of bug reports compared with common documents. Especially priority and severity, which is one of the metadata, are important because they are used to bug report triage. Due to the triage process, developers can be informed prior to the processing of important and critical bug reports. Figures 2 and 3 show examples of bug reports. Bug reports in Bugzilla are known to address a substantial amount of metadata. Bugzilla even supports "importance (priority and severity)" and "triage owner", which are related to the triage process, and common data such as "reporter", "product" and "status." Figure 2 shows a bug report in Bugzilla. The report presents an unlimited page loading and information leaks. The priority "p2" in the bug report enable developers to fix the bug as soon as possible before reading it closely (Bugzilla uses stages p1–p5 as priority, where p1 is the highest priority). Bug reports in Github have a substantial amount of information about the environment in which the bug appears. A bug report, e.g., "enhancement", "discussion", and "question", is usually not uploaded. Figure 3 shows a bug report in Github. The bug report shows cases in which a segmentation fault related with disconnected monitors. The bug occurred in iOS version 12, and the report describes the environment in which the bug occurs by showing the code. This study uses bug reports from Bugzilla and MSR that support bug reports in Fire Fox and Eclipse.


Open Bug 1511914 Opened last year Updated 10 days ago

Style Editor does not cache repeated loads of the same file, allowing unlimited memory use

▾ Categories

Product: DevTools ▾
Component: Style Editor ▾
Type: defect
Priority: P2

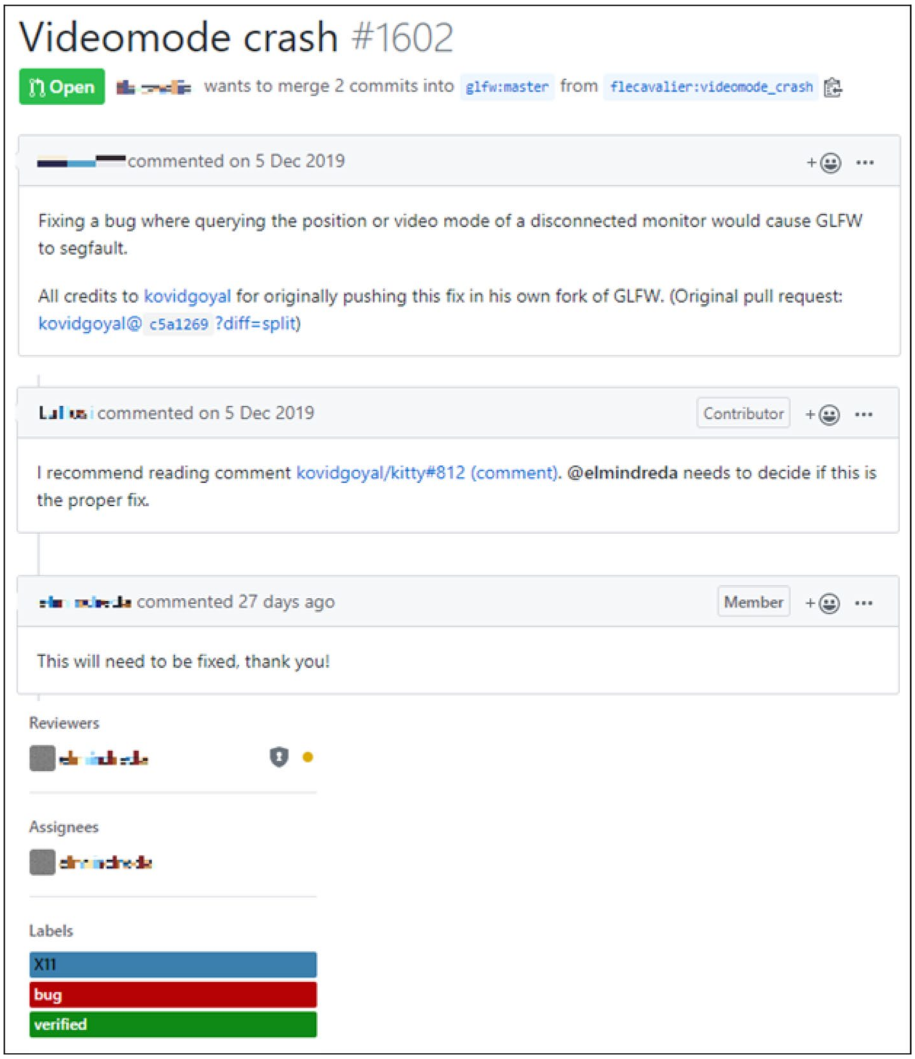
[redacted] Reporter
Description • Last year

Attached file memory.html — Details

```
<!--
    Use Firefox's Style Editor to read local files anc
-->
<html>
    <head>
    <script>
        // Load multiple copies of a file.
        const NUM_COPIES = 3000;

        // Load a stylesheet from the filesystem
        const addStyleSheet = (path) => {
            e = document.createElement('link');
```

```
The devtools Style Editor loads file:// URLs from disk. A web page can add files programmatically, and the
Style Editor will display them once it is active.

# Memory
Opening the Style Editor tab can consume gigabytes of memory if a page repeatedly loads a large file as a
stylesheet. This doesn't trigger Firefox's "A web page is slowing down your browser" warning.

This issue leads to exhaustion of system memory on Linux, and causes the Firefox tab or parent process to crash
on Windows.

# Information leaks
Reading local files could be a problem for kiosks. If the style editor's manual upload is patched out,
JavaScript can still load content from disk.

This might also cause trouble if the devtools Extension API adds the Style Editor, and files can be read and
stolen that way.

# Notes
Tested with Firefox 63.0.3 on Ubuntu 18.04 and Windows 10.

On Ubuntu 18.04, /lib and /etc/passwd can be loaded from an http:// or https:// page, but sometimes /usr and
other paths cannot. C: works on Windows.
```

Fig. 2 Bug report in Bugzilla [74]



Videomode crash #1602

Open [redacted] wants to merge 2 commits into `glfw:master` from `flecavalier:videomode_crash`

[redacted] commented on 5 Dec 2019

Fixing a bug where querying the position or video mode of a disconnected monitor would cause GLFW to segfault.

All credits to kovidgoyal for originally pushing this fix in his own fork of GLFW. (Original pull request: kovidgoyal@ `c5a1269` ?diff=split)

[redacted] commented on 5 Dec 2019 Contributor

I recommend reading comment kovidgoyal/kitty#812 (comment). **@elmindreda** needs to decide if this is the proper fix.

[redacted] commented 27 days ago Member

This will need to be fixed, thank you!

Reviewers

[redacted]

Assignees

[redacted]

Labels

X11

bug

verified

Fig. 3 Bug report in GitHub [75]

LDA

LDA is a probability model in which topics exist in each document for the given document collection (corpus). Users can estimate the words distribution by topics and topic distribution by documents using LDA. In LDA, documents consist of topics, and topics generate words based on the probability distribution. LDA traces the back process and creates the document when data are input. T is a topic variable, D is a document variable and W is a word variable. The trace back process is described as follows:.

```
1)  Assign all words in all documents to random topics (of course, most of them
    are wrong).

    Repeat:
    {
      2)  For W, assume that W is misassigned but the other variables are
          correctly assigned.
      3)  Reassign W according to two conditional probabilities as follows:
          -  P(T | D): the distribution of topics in the same document.
          -  P(T | W): the distribution of topics for the same word.
    }
```

Figure 4 shows an example of supposition for generating a document in LDA. If a machine knows the distribution of topics in documents, a document can be generated using supposition of LDA. In a chart of the figure, the distribution from topic 1 to topic 4 is 0.15, 0.2, 0.35 and 0.3. The machine stochastically selects a topic. In the figure, topic 1 is selected with a 15% probability. The machine selects a word that consists of topic 1 (all topics consist of words that are well matched with the topic). In the figure, "basic" is selected. Topic 2 is selected with a 20% probability, and "function" is selected. If the machine repeats this routine, the document is completed.

Figure 5 shows an example of traceback in LDA. LDA builds the distribution of topics by tracing back to the supposition in Fig. 4. First, the machine randomly assigns all

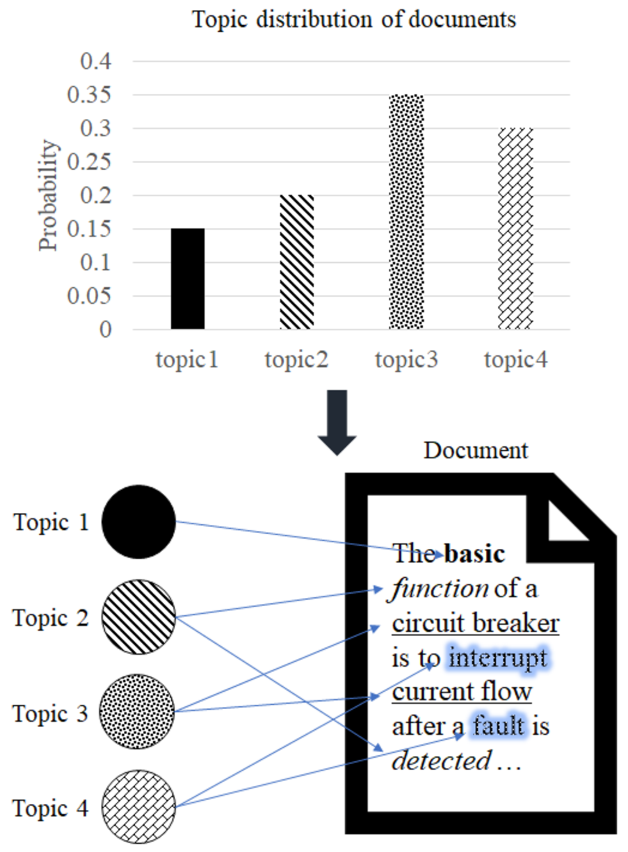

Topic distribution of documents
Probability
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
topic1
topic2
topic3
topic4
Document
Topic 1
Topic 2
Topic 3
Topic 4
The basic function of a circuit breaker is to interrupt current flow after a fault is detected ...

Fig. 4 Example of supposition to generate a document in LDA

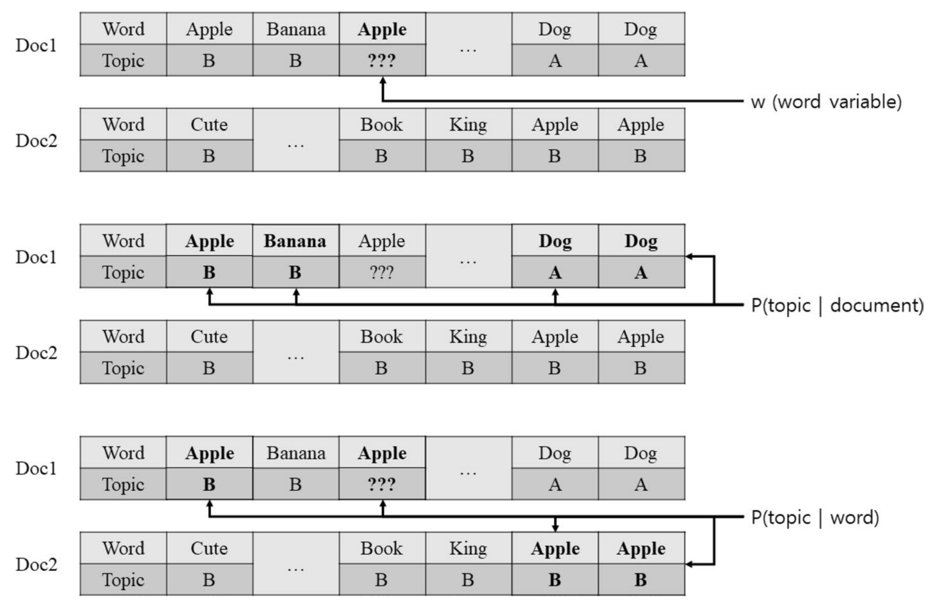


Fig. 5 Bug report in GitHub

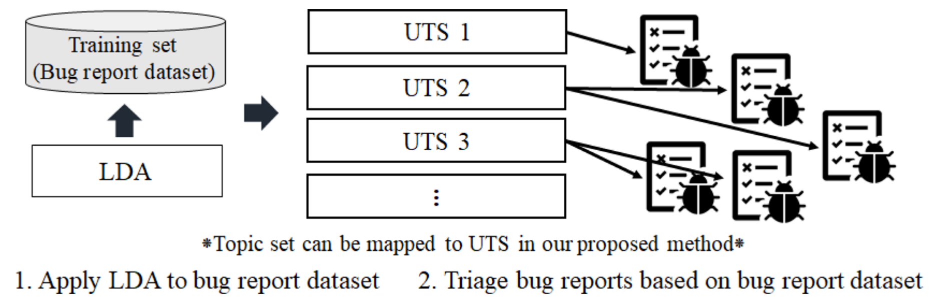

Training set
(Bug report dataset)
LDA
UTS 1
UTS 2
UTS 3
⋮
Topic set can be mapped to UTS in our proposed method
1. Apply LDA to bug report dataset
2. Triage bug reports based on bug report dataset

Fig. 6 Existing LDA classification method

words to topics. As shown in Fig. 5, assume that only two topics exist: A, B and two documents: Doc1, Doc2. For W (third word: "Apple" in Doc1), the machine determines the distribution of topics in the same document (P(T| Doc 1)). Because both A and B appear at 50% in Doc 1, the topic of W cannot be determined.

The machine determines the distribution of the topics for the same word (P(T| "Apple")). In this figure, it obtains the distribution of "Apple" in Doc 1, 2. Because the distribution of B is larger, it determines that the topic of W is B. This study aims to improve triage accuracy and be compatible with state-of-the-art studies that employ multiple LDA.

Approach

This section describes how the proposed method is processed. Figure 6 shows an existing bug report triage process using LDA. Figure 7 demonstrates the overall approach used with the proposed method.

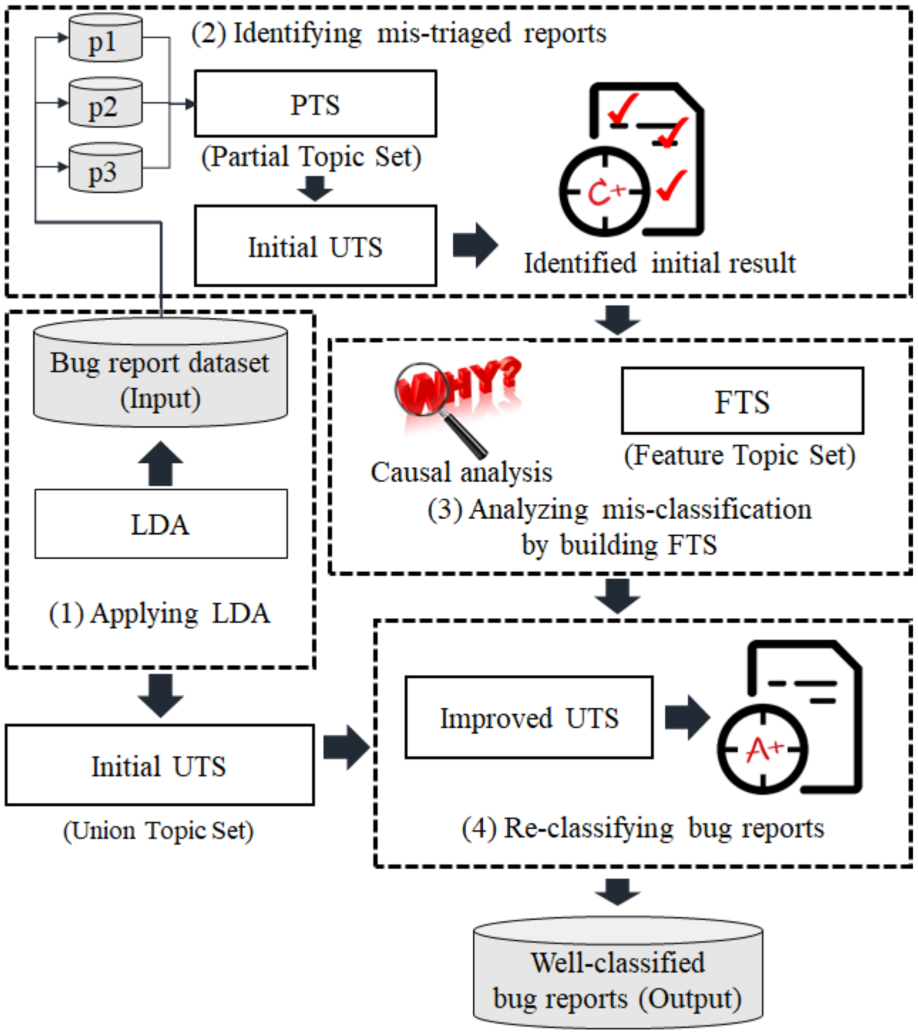

(2) Identifying mis-triaged reports
p1
p2
p3
PTS
(Partial Topic Set)
Initial UTS
Identified initial result
Bug report dataset (Input)
LDA
(1) Applying LDA
WHY?
Causal analysis
FTS
(Feature Topic Set)
(3) Analyzing mis-classification by building FTS
Initial UTS
(Union Topic Set)
Improved UTS
(4) Re-classifying bug reports
Well-classified bug reports (Output)

Fig. 7 Overall approach

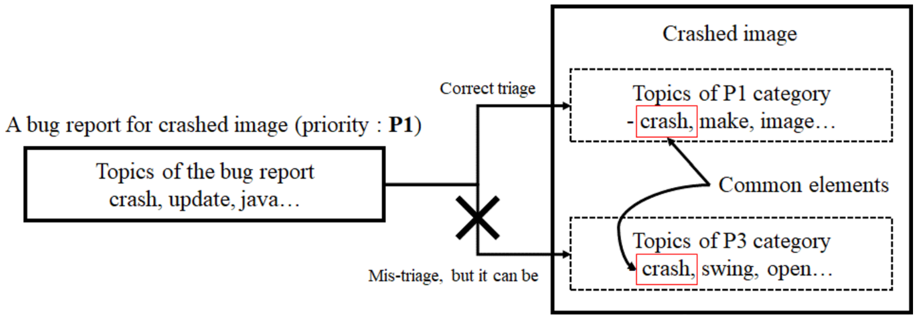


Fig. 8 Example of the mis-triage caused by common elements

Applying LDA to bug report classification

The existing bug report classification applies LDA for a bug report base (dataset), and the machine classifies the bug reports based on the topic sets as the result (a union topic set (UTS) for distinguishing other topic sets that subsequently appear). This process is a part of the proposed method. The existing method is suitable for textual classification but achieves a poor triage performance, one of the reasons for which is the common elements occurring in different topic sets. Figure 8 shows an example of a bug report mis-triage. “Crashed image” is a bug in which an image is not displayed on the page where it should be. A bug report for a crashed image caused by an incorrect extension or an image loader error should be triaged as priority “P1.” Two priorities exist, namely, P1 and P3, in the correct triage model (the topics are listed in order of their influence). In the figure, “crash” is the first to appear in both P1 and P3. Unfortunately, the situation is the same in the bug report. Thus, the machine will apply a triage using topics with a low influence, and even minor errors will cause a mis-triage.

Identifying mis-triaged bug reports

To improve the UTS, including the common elements, the proposed method builds additional topic sets. One set is a partial topic set (PTS). The existing LDA classification cannot determine the priority or severity of the UTS. Thus, it should identify them along with the mis-triaged bug reports. The PTS assumes this role. The PTS-building process is similar to the case of the UTS. The proposed method classifies the bug reports in the training set based on the priority and severity. The PTS representing each field is obtained by applying the field. Figure 9 visualizes the building of the PTS and the process of identifying mis-triaged bug reports. The most popular field in the UTS can be determined by comparing the UTS with the PTS. The method also estimates that bug reports inconsistent with the most popular field will be mis-triaged. The common element problem can be resolved by correctly reclassifying mis-triaged bug reports based on the PTS; however, this method has a particular problem in that it only uses the UTS for bug report classification. This method should round the UTS and PTS for all fields. From a temporal aspect, this step is absolutely inefficient. Thus, the search space of the bug triage should be reduced. This problem is resolved through the next step.

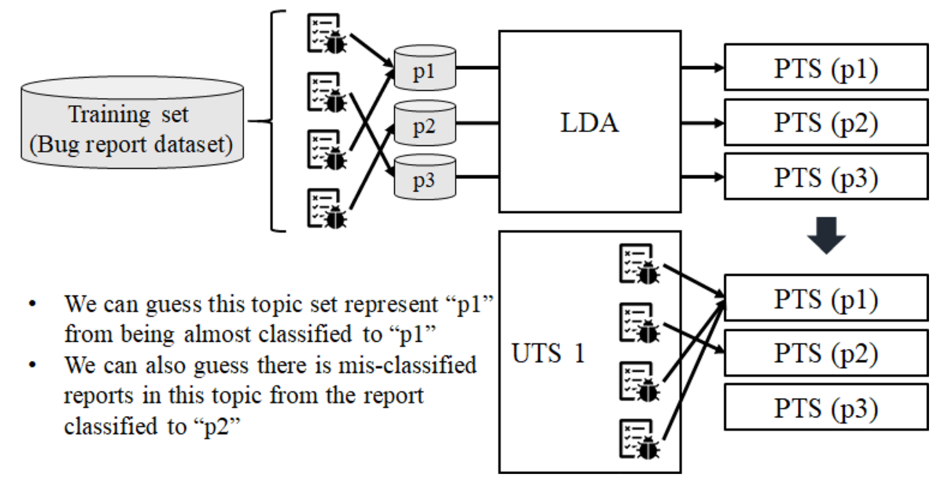

Training set
(Bug report dataset)
p1
p2
p3
LDA
PTS (p1)
PTS (p2)
PTS (p3)
UTS 1
PTS (p1)
PTS (p2)
PTS (p3)
• We can guess this topic set represent “p1” from being almost classified to “p1”
• We can also guess there is mis-classified reports in this topic from the report classified to “p2”

Fig. 9 Process for building PTS and identification of mis-triaged reports

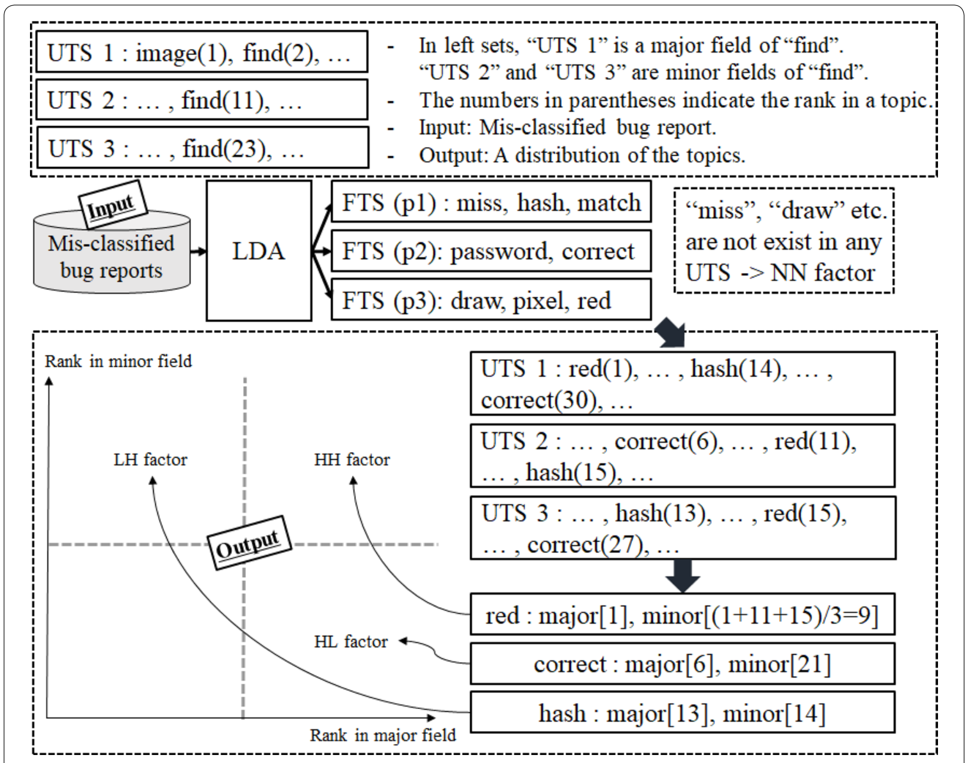

UTS 1 : image(1), find(2), …
UTS 2 : … , find(11), …
UTS 3 : … , find(23), …
- In left sets, "UTS 1" is a major field of "find". "UTS 2" and "UTS 3" are minor fields of "find".
- The numbers in parentheses indicate the rank in a topic.
- Input: Mis-classified bug report.
- Output: A distribution of the topics.
Input
Mis-classified bug reports
LDA
FTS (p1) : miss, hash, match
FTS (p2): password, correct
FTS (p3): draw, pixel, red
"miss", "draw" etc. are not exist in any UTS -> NN factor
Rank in minor field
LH factor
HH factor
Output
HL factor
Rank in major field
UTS 1 : red(1), … , hash(14), … , correct(30), …
UTS 2 : … , correct(6), … , red(11), … , hash(15), …
UTS 3 : … , hash(13), … , red(15), … , correct(27), …
red : major[1], minor[(1+11+15)/3=9]
correct : major[6], minor[21]
hash : major[13], minor[14]

Fig. 10 Process of building FTS and feature of FTS topics

Table 1 Term definitions for analyzing mis-triaged bug reports

Term	Abbreviation	Details
Major field	–	For a topic, a topic set has the highest rank for the UTS, including the topic
Minor field	–	For a topic, all topic sets, with the exception of the major field in the UTS, including the topic.
Number of classifiable bug reports	N-clsf	The number of bug reports that UTSs currently classify
Accuracy of major field	acc-major	The classification accuracy of bug reports in major field
Accuracy of minor field	acc-minor	The classification accuracy of bug reports in minor field
High rank-high rank factor	HH factor	A topic has a high rank for both the major field and the minor field
High rank-low rank factor	HL factor	A topic has a high rank on the major field and a low rank on the minor field.
Low rank-high rank factor	LH factor	A topic has a low rank for the major field and a high rank for the minor field
No rank-no rank factor	NN factor	A topic does not exist in the UTS

Analyzing mis-triaged bug reports by building a feature topic set

To overcoming the temporal limit of the method using the PTS, in this study, a method for reducing the search space of a bug report triage using the feature topic set (FTS) is proposed. This method does not round all PTSs but does round the FTS as the corresponding common elements for the bug reports. Figure 10 shows the analysis process of building the FTS and its features. In the initial results, the proposed method collects mis-triaged bug reports based on the PTS and obtains the FTS by applying LDA to them. The FTS can be constructed in two ways. The first way is to build for the correct destination of mis-triaged bug reports, and the second is to build for the current location and correct destination of the bug report. The latter has the advantage of requiring a smaller search space by designating the current location. However, this approach should be employed when the massive size of the training set is prepared because the number of topics decreases for each FTS. The topics of the FTS are divided into four parts based on the ranking of the UTS. Table 1 shows the terms used in this step.

In Fig. 10, the FTS has several topics; in particular, "red," "correct," and "hash" appear in the original UTS numerous times. The proposed method obtains the ranks of the topics in both the major and minor fields (and determines the average ranks for two or more minor fields). The method selects the topic and corresponding factor based on these ranks. The words that do not appear in the UTS are classified as the NN factor. There are two methods that address common elements. With the first method, common elements are removed from its minor field. As a factor of the common elements, the performance of the UTS is influenced but N-clsf is commonly decreased. If none of the words in a bug report are located in the UTS, the proposed method cannot classify the bug report, i.e., a common element deletion is inefficient. If common elements are included in the HH factor, acc-major improves but acc-minor decreases. If common elements are included in the HL factor, acc-major increases but acc-minor is less effective. As the worst case, if common elements are included in the HH factor, acc-major is unaffected, and acc-minor declines.

The other method that addresses common elements is to disregard them and reclassify the bug report by matching the FTS when the bug report is classified by common elements. Even if this approach increases the temporal cost of classifying bug reports by the FTS, it can prevent a decrease in N-clsf when common elements are removed. In particular, when the NN factor is employed, the process is quick because it does not check common elements in the FTS, unlike other factors. This study uses this method.

Re-classifying bug reports by improving UTS using FTS

To improve the UTS using the FTS, the proposed method builds a factor parser consisting of common elements in the UTS. This method quickly searches using a Hash or Trie (because all common elements are words). A factor parser obtains the addresses of the FTS that correspond to a common element. Figure 11 shows an example in which the UTS is improved using the FTS. The proposed method classifies the bug reports through the UTS, similar to the existing LDA classification. If the bug report includes common elements (particularly HH, HL, and LH factors), it calls the FTS that corresponds to the common elements identified by the factor parser. The NN factor only exists in the FTS. We know that other factors are identified, and the NN factor better represents its major fields. Thus, constructing the FTS using only an NN factor builds a more accurate and quicker environment. The method compares the report with the FTS to check whether the classification is correct.

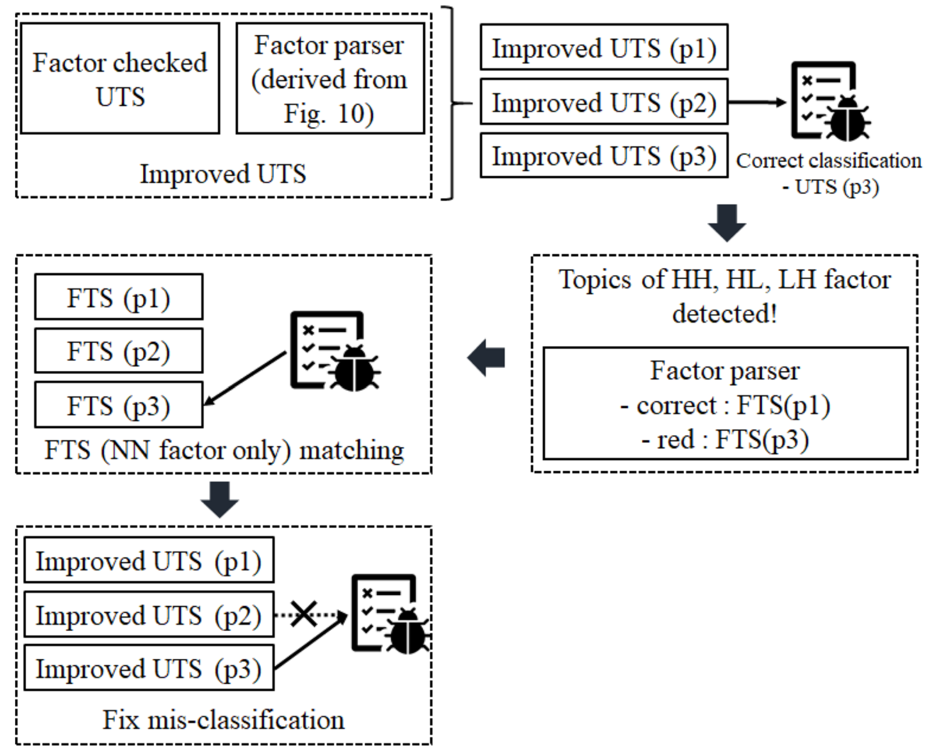


Fig. 11 Example of improving UTS using FTS

Evaluation

Experiment design

To develop an experiment verifying the proposed method, we collected multiple bug reports. In this study, our dataset consists of 3362 bug reports from Bugzilla, which supports various types of bug reports and metadata, and 41,229 bug reports from the MSR, which supports numerous types of bug reports. The bug reports from Bugzilla are classified based on the priority and severity. The bug reports from Bugzilla support both the severity and priority as the metadata that are useful for triage, whereas those from MSR support only the priority for the triage. To conduct the practical experiments, a total of 231 bug reports from Bugzilla related with Git were actually used in the group development. Git is an open-source distributed version-control system for tracking changes in the source code during software development [76]. In this study, our dataset consists of bug reports from Bugzilla that support various bug reports and metadata and bug reports from MSR that support numerous bug reports. The bug reports from Bugzilla are classified by priority and severity.

Thus, we fit a model verifying the improvement in the UTS accuracy of the proposed method. The collected bug reports were divided into three groups: Bugzilla, MSR, and a combination of the two. To build the training and test sets, we employ tenfold cross-validation. In this experiment, we implement the proposed method in Python 3. Python 3 supports various libraries for NLP and topic modeling. We use nltk (NLP), stop-words, and genism (topic modeling).

Experiment results

Table 2 shows the number of bug reports classified by the proposed method in terms of percentage of each fold. The fold represents the divided parts in the cross-validation. "Bugzilla," "MSR," "Bugzilla (Git)," and "Integrated" represent the bug report datasets. The percentages indicate the probability that the bug reports will be classified in each dataset. Bugzilla (Git) consists of bug reports related with Git in Bugzilla dataset. In the "Integrated" dataset "Bugzilla" and "MSR" are combined. As shown in Table 2, the method achieves a classification rate of greater than 97% for all folds for the Bugzilla reports; the average rate is 98.311%. In the case of the MSR, the classification rate is lower than that of Bugzilla, and the average rate is 94.993. In the case of Bugzilla (Git), the classification rate is lower than the entire Bugzilla dataset, such as the MSR, and the average rate is 93.91%. In the case of an integrated environment, the average rate is 96.199%.

Table 3 shows the percentage of each fold for the accuracy of the bug reports classified based on the severity using the LDA in Bugzilla. Table 4 shows the percentage of each fold for the accuracy of the bug reports classified based on the severity when using the proposed method in Bugzilla. The first row of Tables 3 and 4 represents the severity level of the bug reports. The numbers in Tables 3 and 4 denote the accuracy of the

Table 2 Percentage of bug reports classified by the proposed method (%)

Fold	Bugzilla	MSR	Bugzilla (Git)	Integrated
1	98.02	94.26	86.95	97.00
2	97.96	93.73	91.30	97.26
3	98.72	94.03	95.65	96.73
4	98.34	96.66	95.65	95.76
5	99.42	96.00	100.00	95.50
6	97.64	95.13	91.30	96.26
7	98.40	95.80	100.00	94.76
8	98.31	93.26	95.65	95.26
9	97.26	96.23	86.95	95.83
10	99.04	94.83	95.65	97.63
Average	98.31	94.99	93.91	96.20

Table 3 Triage accuracy of bug report classified by LDA (Severity) (%)

Fold	Block	Critical	Major	Normal	Minor	Trivial	Enhancement
1	*50.05*	64.66	55.43	75.55	54.75	45.24	66.35
2	(32.79)	(57.62)	(52.75)	73.96	55.21	55.91	64.18
3	43.11	62.66	*65.39*	67.3	65.71	50.21	68.29
4	37.41	65.62	58.39	*82.59*	55.46	*56.16*	(51.1)
5	45.08	72.43	63.04	74.79	(51.96)	54.03	68.39
6	33.52	*73.54*	56.92	75.93	*69.12*	51.23	62.34
7	38.01	60.94	59.18	78.92	63.61	49.09	66.60
8	35.08	61.10	57.02	70.46	55.68	40.11	54.46
9	44.38	72.27	60.17	(65.46)	60.36	49.92	56.70
10	37.28	68.70	62.85	66.28	59.06	(39.45)	*68.67*
Average	39.67	65.95	59.11	73.12	59.09	49.14	62.71

The maximum value in each column is presented in italics

Table 4 Triage accuracy of bug report classified by the proposed method (Severity)

	Block	Critical	Fold	Normal	Minor	Trivial	Enhancement
1	*81.53*	88.83	83.00	90.70	81.79	78.76	84.85
2	(68.77)	82.27	(78.67)	91.34	82.46	82.94	84.21
3	78.16	86.28	87.39	89.75	87.55	81.21	88.22
4	75.29	88.12	84.43	91.63	81.31	*83.73*	(79.97)
5	77.49	90.74	*89.53*	*93.00*	(80.71)	81.50	89.88
6	71.54	*91.98*	81.50	90.83	*87.71*	80.32	82.62
7	72.33	(82.11)	85.32	91.66	85.29	79.50	87.77
8	70.04	85.55	83.16	90.13	83.09	74.12	82.84
9	78.06	87.55	85.07	89.53	85.20	81.31	83.99
10	73.48	89.40	86.76	(88.16)	84.37	(73.86)	*90.32*
Average	74.67	87.28	84.48	90.67	83.95	79.73	85.47

The maximum value in each column is presented in italics

bug report classification for each severity level. The numbers in italics represent the maximum value, and the numbers in the round brackets represent the minimum value. The proposed method achieves an improvement in accuracy of 25% compared with the original LDA for seven severity fields. In particular, "Block" accounts for 35%. The proposed method shows a classification accuracy of 85% for all fields, with the exception of "Block." The reason for the low effectiveness for "Block" is the small number of bug reports. Because overlapped contexts are rare compared with other fields, the method cannot obtain high weight topics.

Table 5 shows the percentage of each fold for the accuracy of the bug reports classified based on the priority using the LDA. Table 6 shows the percentage of each fold for the accuracy of the bug reports classified based on the priority when using the proposed method. The first row of Tables 5 and 6 represents the accuracy of the bug report classification based on the priority.

Table 5 Triage accuracy of bug report classified by LDA (priority) (%)

Fold	P1	P2	P3	P4	P5
1	64.76	(52.69)	69.50	61.38	66.73
2	60.52	63.07	*76.09*	52.24	70.14
3	60.52	61.38	67.75	*66.57*	62.46
4	60.39	52.79	64.66	56.77	66.60
5	63.23	63.67	(61.35)	(49.16)	(59.12)
6	(49.60)	*66.09*	61.51	54.03	67.74
7	*66.79*	60.04	70.77	56.89	69.12
8	55.11	59.60	62.46	56.89	59.22
9	64.34	59.18	66.38	55.96	69.25
10	62.05	56.32	75.49	65.07	*75.71*
Average	60.73	59.48	67.60	57.50	66.61

The maximum value in each column is presented in italics

Table 6 Triage accuracy of bug report classified by the proposed method (priority) (%)

Fold	P1	P2	P3	P4	P5
1	*88.86*	(82.49)	90.61	86.66	89.84
2	85.39	87.68	*92.22*	82.11	90.70
3	85.83	89.46	89.56	88.44	87.87
4	86.59	83.06	88.00	83.25	90.23
5	86.66	87.49	87.62	(77.97)	84.69
6	(77.78)	88.98	88.92	83.25	88.22
7	86.09	*89.97*	90.89	84.46	89.94
8	82.33	85.16	(85.93)	85.55	(84.08)
9	86.91	83.09	87.39	85.51	90.80
10	84.97	83.48	91.47	*89.62*	*91.05*
Average	85.14	86.09	89.26	84.68	88.74

The maximum value in each column is presented in italics

The proposed method achieves an improvement in accuracy of 24% compared with the original LDA for five priority fields. In particular, the method achieves an excellent performance for "P2" and "P4."

Discussion

Compatibility with original LDA in the existing studies

This section discusses how the proposed method is compatible (substitutable) with the LDA for the existing combined methods (the LDA with other methods). Zou et al. [62] defined two constraints used in generating a combined method. First, the base techniques should apply the same information source. If they use different sources of information, it is necessary to conduct a data conversion. When the user cannot develop a data converter, this combined method is impossible to correctly build. Even if a data converter is developed, it can create other defects. Second, the correlation should be low between the base techniques used for the combination. Zou et al. [77] categorized fault-localization (FL) techniques into seven FL families. They demonstrated a set of techniques with a weak correlation with each other.

Related to the first condition, the proposed method uses bug report dataset that is the same information source with LDA. Related to the second condition, the existing studies used a combination of low correlation techniques with LDA. That is, the proposed method also ensure low correlation with the techniques used in the existing studies because this method is based on the multiple LDA and the same information source with LDA mentioned above. Thus, the proposed method can be used as an alternative to original LDA in the combined LDA methods of the existing studies.

Statistical comparison with the proposed method and original LDA

Table 7 shows the results for the paired T-test of each field in Tables 3 and 4. A paired t-test is used to compare two population means for two samples, which is generally used before-and-after observations on the same subjects [78, 79]. Table 8 shows the results for the paired T-test of each field in Tables 5 and 6. The T-statistic is a value obtained by the T-test for each field. The P-value is a statistical value used to compare the proposed

Table 7 Paired T-test results for severity accuracy

Severity	P-value (the method vs LDA)	T-statistic	Pearson's r
Block	2.00E−12	51.77	0.95
Critical	5.13E−09	21.33	0.86
Major	2.19E−12	50.80	0.91
Normal	8.79E−07	11.81	0.76
Minor	1.44E−09	24.60	0.96
Trivial	6.57E−11	34.81	0.97
Enhancement	1.71E−08	18.61	0.86

Table 8 Paired T-test results for priority accuracy

Priority	P-value (the method vs LDA)	T-statistic	Pearson's r
P1	2.00E−10	30.12	0.91
P2	2.53E−10	29.93	0.78
P3	2.00E−08	18.28	0.87
P4	1.09E−10	32.87	0.93
P5	3.12E−09	22.56	0.89

method and the original LDA. The null hypothesis (H_0) indicates that no statistically significant difference exists between the proposed method and the original LDA. The alternative hypothesis (H_1) shows that a statistically significant difference does exist between the proposed method and the original LDA. In Tables 7 and 8, all p-values are less than 0.05. We reject H_0 and adopt H_1. Thus, the proposed method has a statistically significant difference compared with the original LDA, i.e., the proposed method is better than the original LDA classification.

Conclusions

In this paper, a method for improving LDA, which is generally employed in a bug report triage, was proposed. To improve the classification accuracy of the topic set used with LDA, the proposed method builds additional topic sets and improves the original set. To validate the proposed method, we used bug report platforms applied in practice, namely, Bugzilla, MSR, Bugzilla (Git), and an integrated platform. We demonstrated how the method accurately classifies the bug reports using the experiment results, and how it is better than the original LDA based on a statistical paired T-test.

Traditional bug triage methods try to cover their weakness by combining the LDA with other methods. However, the combined hybrid methods may have compatibility problems, such as a correlation or difference in the input data applied. Because the proposed method focuses on upgrading the LDA itself, such compatibility issues do not occur. In addition, the proposed method will provide the basis for the development of more improved hybrid methods.

Even if we improve triage accuracy via this study we will perform further research on the following issues. Although many related studies use bug reports to help fix bugs, studies that address fixing bugs using comments are lacking. We intend to study the relation between comments and bug reports and identify bugs based on this relation. We also will study the automation of bug identification using the results of these studies.

Abbreviations
LDA: Latent Dirichlet Allocation; QA: Quality Assurance; KNN: K-Nearest Neighbors; SVM: Support Vector Machine; MSR: Mining Software Repositories; IR: Information Retrieval; RTM: Relational Topic Model; HMM: Hidden Markov Model; CoLUA: Configuration Bug Learner Uncovers Approved options; NLP: Natural Language Processing; KSAP: K-NN search and heterogeneous proximity; UTS: Union Topic Set; PTS: Partial Topic Set; FTS: Feature Topic Set; FL: Fault Localization.

Acknowledgements
Not applicable.

Authors' contributions
All the authors review the final manuscript for submission. Both authors read and approved the final manuscript.

Funding
This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2017R1C1B5018295).

Availability of data and materials
The datasets used during the current study are available from the corresponding author on reasonable request.

Competing interests
The authors declare that they have no competing interests.

Received: 28 September 2019 Accepted: 25 April 2020
Published online: 16 June 2020

References
1. Tunio MZ, Luo H, Wang C, Zhao F (2018) Crowdsourcing software development: task assignment using PDDL artificial intelligence planning. J Inf Processing Syst 14(1):129–139
2. Park JH, Salim MM, Jo JH, Sicato JCS, Rathore S, Park JH (2019) CIoT-Net: a scalable cognitive IoT based smart city network architecture. Hum Centric Comput Inf Sci 9(1):29
3. Jang Y, Park CH, Seo YS (2019) Fake news analysis modeling using quote retweet. Electronics 8(12):1–20
4. Kim SW, Gil JM (2019) Research paper classification systems based on TF-IDF and LDA schemes. Hum Centric Comput Inf Sci 9(1):30
5. Tian Y, Song W, Sun S, Fong S, Zou S (2019) 3D object recognition method with multiple feature extraction from LiDAR point clouds. J Supercomput 75(8):4430–4442
6. Song W, Tian Y, Fong S, Cho K, Wang W, Zhang W (2016) GPU-accelerated foreground segmentation and labeling for real-time video surveillance. Sustainability 8(10):916
7. Huh JH, Seo YS (2019) Understanding edge computing: engineering evolution with artificial intelligence. IEEE Access 7:164229–164245
8. Seo YS, Huh JH (2019) Automatic emotion-based music classification for supporting intelligent IoT applications. Electronics 8(2):1–20
9. Wang J, Ju C, Gao Y, Sangaiah AK, Kim GJ (2018) A PSO based energy efficient coverage control algorithm for wireless sensor networks. Comput Mater Contin 56(3):433–446
10. Wang J, Gao Y, Liu W, Sangaiah AK, Kim HJ (2019) An intelligent data gathering schema with data fusion supported for mobile sink in wireless sensor networks. Int J Distrib Sens Netw 15(3):1–9
11. Wang J, Gao Y, Yin X, Li F, Kim HJ (2018) An enhanced PEGASIS algorithm with mobile sink support for wireless sensor networks. Wirel Commun Mobile Comput 2018:1–9
12. Wang J, Wu W, Liao Z, Sangaiah AK, Sherratt RS (2019) An energy-efficient offloading scheme for low latency in collaborative edge computing. IEEE Access 7:149182–149190
13. Jimoh RG, Balogun AO, Bajeh AO, Ajayi S (2018) A PROMETHEE based evaluation of software defect predictors. J Comput Sci Appl 25(1):106–119
14. Laradji IH, Alshayeb M, Ghouti L (2015) Software defect prediction using ensemble learning on selected features. Inf Softw Technol 58:388–402
15. Tran HM, Le ST, Nguyen SV, Ho PT (2020) An analysis of software bug reports using machine learning techniques. SN Comput Sci 1(1):4
16. García-Floriano A, López-Martín C, Yáñez-Márquez C, Abran A (2018) Support vector regression for predicting software enhancement effort. Inf Softw Technol 97:99–109
17. Alaqail H, Ahmed S (2018) Overview of software testing standard ISO/IEC/IEEE 29119. Inf Softw Technol 18(2):112–116

18. Mann M, Tomar P, Sangwan OP (2018) Bio-inspired metaheuristics: evolving and prioritizing software test data. Appl Intell 48(3):687–702
19. Zhang H (2019) Research on software development and test environment automation based on android platform. 3rd International Conference on mechatronics engineering and information technology. Atlantis Press, Paris
20. Thakur D, Types of software maintenance. http://ecomputernotes.com/software-engineering/types-of-software-maintenance. Accessed 22 Sep 2019
21. Stojanov Z, Stojanov J, Dobrilovic D, Petrov N (2017) Trends in software maintenance tasks distribution among programmers: A study in a micro software company. 2017 IEEE 15th International Symposium on intelligent systems and informatics, pp 23–28
22. Jang JW (2018) Improvement of the automobile control software testing process using a test maturity model. J Inf Process Syst 14(3):607–620
23. Life Cycle of a bug. https://www.bugzilla.org/docs/2.18/html/lifecycle.html
24. Anvik J, Hiew L, Murphy GC (2005) Coping with an open bug repository. Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange—eclipse '05. pp 35–39
25. Ye X, Fang F, Wu J, Bunescu R, Liu C (2018) Bug Report Classification using LSTM architecture for more accurate software defect locating. 17th IEEE International Conference on machine learning and applications. pp 1438–1445
26. Terdchanakul P, Hata H, Phannachitta P, Matsumoto K (2017) Bug or not? bug report classification using N-gram IDF. IEEE International Conference on software maintenance and evolution. pp 534–538
27. Guo S, Chen R, Wei M, Li H, Liu Y (2018) Ensemble data reduction techniques and multi-RSMOTE via fuzzy integral for bug report classification. IEEE Access 6:45934–45950
28. Kukkar A, Mohana R (2018) A supervised bug report classification with incorporate and textual field knowledge. Procedia Comput Sci 132:352–361
29. Du X, Zheng Z, Xiao G, Yin B (2017) The automatic classification of fault trigger based bug report. IEEE International Symposium on software reliability engineering workshops. pp 259–265
30. Xu R, Ye L, Xu J (2013) Reader's emotion prediction based on weighted Latent Dirichlet Allocation and multi-label k-nearest neighbor model. J Comput Inf Syst 9(6):2209–2216
31. Safi'ie MA, Utami E, Fatta HA (2018) Latent Dirichlet Allocation (LDA) model and knn algorithm to classify research project selection. IOP Conference Series Mater Sci Engin 333(1):012110
32. Chen W, Zhang X (2017) Research on text categorization model based on LDA—KNN. 2017 IEEE 2nd advanced information technology, electronic and automation Control Conference. pp 2719–2726
33. Liu X, Agarwal S, Ding C, Yu Q (2016) An LDA-SVM active learning framework for web service classification. 2016 IEEE International Conference on web services. pp 49–56
34. Wang X, Wang J, Yang Y, Duan J (2017) Labeled LDA-Kernel SVM: A short Chinese text supervised classification based on sina weibo. 2017 4th International Conference on information science and control engineering. pp 428–432
35. Deliu I, Leichter C, Franke K (2018) Collecting cyber threat intelligence from hacker forums via a two-stage, hybrid process using support vector machines and Latent Dirichlet Allocation. 2018 IEEE International Conference on Big Data. pp 5008–5013
36. Lee DG, Seo YS (2019) Systematic review of bug report processing techniques to improve software management performance. J Inf Processing Syst. 15(4):967–985
37. Bugzilla. https://bugzilla.mozilla.org/home. Accessed 22 Sep 2019
38. Mining challenge. http://2012.msrconf.org/challenge.php#challenge_data. Accessed 22 Sep 2019
39. Martie L, Palepu VK, Sajnani H, Lopes C (2012) Trendy bugs: topic trends in the android bug reports. In Proc. MSR. pp 120–123
40. Alipour A, Hindle A, Stroulia E (2013) A contextual approach towards more accurate duplicate bug report detection. Proceeding MSR '13 Proceedings of the 10th Working Conference on mining software repositories. pp 183–192
41. Hindle A, Alipour A, Stroulia E (2016) A contextual approach towards more accurate duplicate bug report detection and ranking. Empir Softw Eng 21(2):368–410
42. Guana V, Rocha F, Hindle A, Stroulia E (2012) Do the stars align? multidimensional analysis of android's layered architecture. Mining Software Repositories (MSR) 2012 9th IEEE Working Conference on. IEEE, New York, pp 124–127
43. Hindle A, Ernst NA, Godfrey MW. Mylopoulos J (2011) Automated topic naming to support cross-project analysis of software maintenance activities. Proceedings of the 8th Working Conference on mining software repositories. ACM. pp 163–172
44. Han D, Zhang C, Fan X, Hindle A, Wong K, Stroulia E (2012) Understanding android fragmentation with topic analysis of vendorspecific bugs. 19th Working Conference on reverse engineering. pp 83–92
45. Sun C, Lo D, Khoo S, Jiang J (2011) Towards more accurate retrieval of duplicate bug reports. Proceedings of the 2011 26th IEEE/ACM International Conference on automated software engineering. IEEE Computer Society. pp 253–262
46. Budhiraja A, Dutta K, Shrivastava M, Reddy R (2018) Towards Word Embeddings for Improved Duplicate Bug Report Retrieval in Software Repositories. Proceedings of the 2018 ACM SIGIR International Conference on theory of information retrieval. pp 167–170
47. Aggarwal K, Rutgers T, Timbers F, Hindle A, Greiner R, Stroulia E (2015) Detecting duplicate bug reports with software engineering domain knowledge. In: SANER 2015: International Conference on software analysis, evolution and reengineering. pp 211–220
48. Aggarwal K, Timbers F, Rutgers T, Hindle A, Stroulia E, Greiner R (2017) Detecting duplicate bug reports with software engineering domain knowledge. J Softw Evol Process 29(3):e1821
49. Campbell JC, Santos EA, Hindle A (2016) The unreasonable effectiveness of traditional information retrieval in crash report deduplication. 2016 IEEE/ACM 13th Working Conference on mining software repositories (MSR). pp 269–280
50. Hindle A, Onuczko C (2019) Preventing duplicate bug reports by continuously querying bug reports. Empir Softw Eng. 24(2):902–936
51. Nguyen AT, Nguyen TT, Nguyen TN, Lo D, Sun C (2012) Duplicate bug report detection with a combination of information retrieval and topic modeling. Proc. ASE'12. pp 70–79

52. Chang J, Blei DM (2009) Relational topic models for document networks, In AIStats. pp 81–88
53. Tian Y, Sun C, Lo D (2012) Improved duplicate bug report identification. In CSMR, 2012. pp 385–390
54. Jalbert N, Weimer W (2008) Automated duplicate detection for bug tracking systems, in dependable systems and networks with FTCS and DCC 2008. DSN 2008. IEEE International Conference on. IEEE, New York. pp 52–61
55. Ebrahimi N, Trabelsi A, Islam MS, Hamou-Lhadj A, Khanmohammadi K (2019) An HMM-based approach for automatic detection and classification of duplicate bug reports. Inf Softw Technol 113:98–109
56. Budhiraja A, Dutta K, Reddy R, Shrivastava M (2018) DWEN: deep word embedding network for duplicate bug report detection in software repositories. Proceedings of the 40th International Conference on software engineering: companion proceeedings. pp 193–194
57. Tamrawi A, Nguyen TT, Al-Kofahi JM, Nguyen TN (2011) Fuzzy-set and cache-based approach for bug triaging. Proc. 19th ACM SIGSOFT Symp. Foundations of software engineering (FSE'11). pp 365–375
58. Wang S, Zhang W, Wang Q (2014) Fixercache: unsupervised caching active developers for diverse bug triage. In ACM/IEEE International Symposium on empirical software engineering and measurement 25
59. Wen W, Yu T, Hayes JH (2016) Colua: Automatically predicting configuration bug reports and extracting configuration options. in 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE). pp 150–161
60. Zhang W, Wang S, Wang Q (2016) KSAP: an approach to bug report assignment using KNN search and heterogeneous proximity. Inf Softw Technol 70:68–84
61. Zhang ML, Zhou ZH (2007) ML-KNN: a lazy learning approach to multi-label learning. Pattern Recogn 40(7):2038–2048
62. Xia X, Lo D, Wang X, Zhou B (2013) Accurate developer recommendation for bug resolution. In WCRE'13. pp 72–81
63. Wu W, Zhang W, Yang Y, Wang Q (2011) DREX: Developer recommendation with k-nearest-neighbor search and expertise ranking. in: APSEC, IEEE, New York, pp 389–396
64. Xie X, Zhang W, Yang Y, Wang Q (2012) DRETOM: developer recommendation based on topic models for bug resolution. In PROMISE'12. pp 19–28
65. Prabhakar RN, Ranjith KS (2016) Effective bug triage with software data reduction techniques using clustering mechanism. i-Manager's J Inf Technol 5(3):15–23
66. Chaudhari RA, Bodake SV (2017) Effective bug triage using software data reduction techniques. Int J Innovative Res Sci Technol 4(1):214–220
67. Kirubakaran S, Maheswari K (2016) Auto-bug triager for assisting manual bug triage. Asian J Inf Technol 15(8):1334–1339
68. Govindasamy V, Akila V, Anjanadevi G, Deepika H, Sivasankari G (2016) Data reduction for bug triage using effective prediction of reduction order techniques. 2016 International Conference on Computation of power, energy information and communication. pp 85–90
69. Sahu K, Lilhore UK, Agarwal N (2018) An improved data reduction technique based on KNN & NB with hybrid selection method for effective software bugs triage. Eng Inf Technol 3(5):1835146
70. Yin Y, Dong X, Xu T (2018) Rapid and efficient bug assignment using ELM for IOT software. IEEE Access 6:52713–52724
71. Florea AC, Anvik J, Andonie R (2017) Spark-based cluster implementation of a bug report assignment recommender system. International Conference on artificial intelligence and soft computing. pp 31–42
72. Florea AC, Anvik J, Andonie R (2017) Parallel implementation of a bug report assignment recommender using deep learning. International Conference on artificial neural networks. pp 64–71
73. Lee SR, Heo MJ, Lee CG, Kim M, Jeong G (2017) Applying deep learning based automatic bug triager to industrial projects. Proceedings of the 2017 11th Joint Meeting on foundations of software engineering. pp 926–931
74. Bug report from Bugzilla. https://bugzilla.mozilla.org/show_bug.cgi?id=1511914. Accessed 30 Jan 2020
75. Bug report from Github. https://github.com/glfw/glfw/pull/1602. Accessed 30 Jan 2020
76. Git. https://git-scm.com/. Accessed 30 Jan 2020
77. Zou D, Liang J, Xiong Y, Ernst MD, Zhang L (2019) An empirical study of fault localization families and their combinations. IEEE Transactions on Software Engineering (Early access)
78. Cleophas TJ, Zwinderman AH (2018) Bayesian paired T-Test. Modern bayesian statistics in clinical research. pp 49–58
79. Seo YS, Bae DH (2013) On the value of outlier elimination on software effort estimation research. Empir Softw Eng 18(4):659–698