Metadata of the chapter that will be visualized in SpringerLink

Book Title	Machine Intelligence, Tools, and Applications	
Series Title		
Chapter Title	Mandelbug Classification Engine: Transfer Learning and NLP Approach	
Copyright Year	2024	
Copyright HolderName	The Author(s), under exclusive license to Springer Nature Switzerland AG	
Corresponding Author	Family Name	**Biswal**
	Particle	
	Given Name	**Biswajit**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	
	Address	300 College Street NE, Orangeburg, SC, 29117, USA
	Email	bbiswaji@scsu.edu

Abstract	Classifying bugs in software systems indeed often involves considering factors like severity, complexity and reproducibility. More elusive and troublesome types of bugs in software development are Mandelbugs which exhibit characteristics of being both complex and non-deterministic, making them exceptionally challenging to reproduce and resolve. However, developers can perform a quick, inexpensive yet most effective methods to identify Mandelbug root causes, and design targeted fault-tolerance mechanisms to enhance system reliability and resilience. His work studied the distribution of Mandelbugs and proposed a classification engine – machine learning, feature engineering, transfer learning and natural language processing (NLP) approach to quickly and effectively categorize Mandelbugs. We evaluated our proposed solution by extracting and processing the text descriptions of Mandelbugs obtained from four different datasets which has 210 Mandelbug records. Our performance evaluation revealed that use of transfer-learning approach has improved F1-scores as well as accuracy (30% - 65%) when compared to that of baseline classifiers.

Keywords (separated by '-')	Machine learning - feature engineering - transfer learning - mandelbugs - natural language processing

Mandelbug Classification Engine: Transfer Learning and NLP Approach

Biswajit Biswal(✉)

300 College Street NE, Orangeburg, SC 29117, USA
bbiswaji@scsu.edu

Abstract. Classifying bugs in software systems indeed often involves considering factors like severity, complexity and reproducibility. More elusive and troublesome types of bugs in software development are Mandelbugs which exhibit characteristics of being both complex and non-deterministic, making them exceptionally challenging to reproduce and resolve. However, developers can perform a quick, inexpensive yet most effective methods to identify Mandelbug root causes, and design targeted fault-tolerance mechanisms to enhance system reliability and resilience. His work studied the distribution of Mandelbugs and proposed a classification engine – machine learning, feature engineering, transfer learning and natural language processing (NLP) approach to quickly and effectively categorize Mandelbugs. We evaluated our proposed solution by extracting and processing the text descriptions of Mandelbugs obtained from four different datasets which has 210 Mandelbug records. Our performance evaluation revealed that use of transfer-learning approach has improved F1-scores as well as accuracy (30% - 65%) when compared to that of baseline classifiers.

Keywords: Machine learning · feature engineering · transfer learning · mandelbugs · natural language processing

1 Introduction

Classification of software system faults are important for critical missions, [1–4] presented four different bug categories such as"Bohrbugs","Heisenbugs","Mandelbugs", and"Schroedinbug"; each bug category represents a different level of complexity and unpredictability, requiring specific approaches and techniques for identification and resolution.

Named after Benoît Mandelbrot, Mandelbugs bugs are characterized by their complexity and unpredictability. Mandelbugs often involve intricate interactions between different components or systems within the software. They exhibit non-linear behavior and may be influenced by a myriad of factors, including concurrency, timing, and environmental conditions. Mandelbugs are notoriously hard to reproduce and may require extensive analysis and debugging to identify and resolve. According to Cotroneo et al. Mandelbugs required more time to fix e.g. 230 h in Linux system and needed peculiar approach to handle it [5]. This motivated us to study Mandelbugs.

S. Dehuri et al. (Eds.): ICMITA 2023, LAIS 40, pp. 1–11, 2024.
https://doi.org/10.1007/978-3-031-65392-6_3

Hard to find, Mandelbugs are classified into "aging related bugs" such as memory leak, cursor leak, TCP aging, numeric overflow, Fragmentation, memory trampler and "non-aging related bugs" such as race condition, lag, overload, limit, timeout, abort, retry, uninitialized bit [6].

Indeed, developers could greatly benefit from tools that aid in for fast and cost-effective bug analysis in assigning categories to bugs. Transfer learning is a solution where models can be trained in one domain and tested with data from another domain under the condition that domains must share some commonalities.

In this work, we propose a natural language processing and transfer learning solution to categorizes Mandelbugs into categories: LAG, ENV, TIM, and SEQ in a fast and cost-effective way. Our objective is to label a Mandelbug into one of those categories based on the Mandelbug text description given in the corresponding bug record. We then evaluated our proposed method on four benchmark datasets [7] such as Mysql-RDBMS, AXIS-Systems, Apache-Server, and Linux-Kernel.

The rest of the sections are arranged as follows. We have described the related works in Sect. 2. Section 3 outlines the overall framework solution. Section 4 narrates Feature Engineering. Transfer learning (TL) is presented in Sect. 5. Experimental results in Sect. 6. Section 7 narrates conclusions and future work.

2 Related Works

2.1 Categorization and Bug Prediction

Numerous studies had been done on bug types prediction [5, 8–12]. Authors of [5] studied software bugs, and concluded that specific strategies required to handle Mandelbugs as it takes longer time to fix. Gegick et al. studied software bugs to determine if a bug was related to security or not using text mining techniques [8]. Arshad et al. developed a tool ConfGauage to address software system configuration based issues such as type of problem, time of problem, problem manifestation and source of problem [9]. Zaman et al. provided a comparative study between performance and security bugs. According to their findings performance bugs needed more files to be modified to fix, while the later can be fixed through extra time [10]. Xia et al. studied external interface bugs while developing softwares applications on different software building tools [11]. Thung et al. studied three bug classes such as structural, nonfunctional, and contral-data flow. Then they developed a method to classify those bug reports automatically [12]. Authors of [13] studied and developed a method to classify configuration bugs using feature selection and data-mining techniques. Our proposed method best aligned with above studies and integrates transfer learning, feature engineering and natural language processing to categorize an unlabeled-bug as a Mandelbug subcategory.

2.2 Natural Language Mining

Text mining studies on software engineering had been reported by many researchers [14–19]. The surveys reported here by us are randomly selected from the scholarly articles published, related to our work and has no bias towards any articled. Wu et al. proposed

a link retrieval technique to detect duplicate bugs and their corresponding changes [14]. Authors of the articles [15–17] had studied and proposed text mining techniques to accurately identify duplicated bug reports. Marcus and Maletic developed an indexing technique named Latent Semantic Indexing (LSI) for duplicate bugs by tracing down links between source code documentations [18]. Haiduc et al. used a text summarization automation process to understand software code succinctly [19]. Zhou et al. developed a relevancy method by taking in a bug report and the returned source code files are then further processed to find a likely input bug report [20].

2.3 Research gaps in studying Mandelbugs

Fig 1, represnts a Mandelbug example in AXIS engine. It describes a bug that causes a concurrency issue with web service implementation class. The main source of this defect was the activation timing of inputs and operations. Occasional occurance of this bug is hard to reproduce. Thus, this bug is categorized as a Mandelbug subcategory TIM [5].

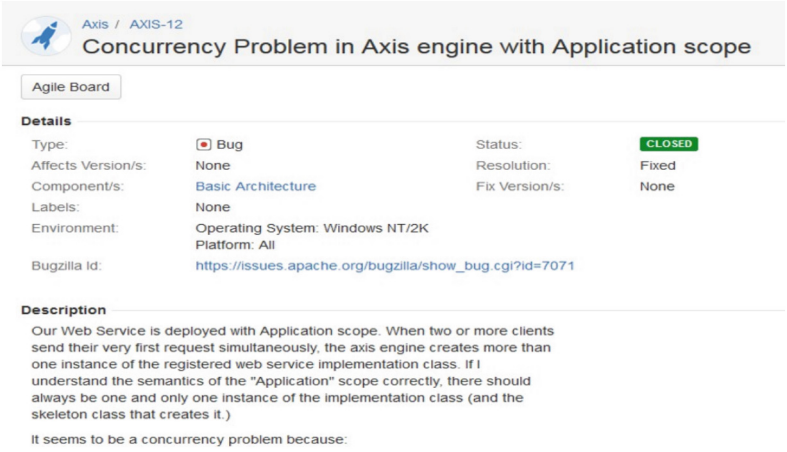

Fig. 1. Mandelbug example with AXIS engine

3 Framework Overview

We present here a feature engineering and transfer learning approach (FETLA) framework as shown in Fig. 2, to categorize Mandelbugs by transferring the knowledge learned from known bugs. Features are extracted from textual data through feature engineering process. Generally in machine learning, training and testing datasets are used alternate terms source domain and target domain respectively. In this research the source domain and the target domain datasets consist of different Mandelbug subcategory records. Thus, in our example the labeled bugs are from source domain, while the unlabeled bugs are from target domain. To differentiate the unlabeled bugs in target domain from labeled bugs in source domain the transfer learning technique is used.

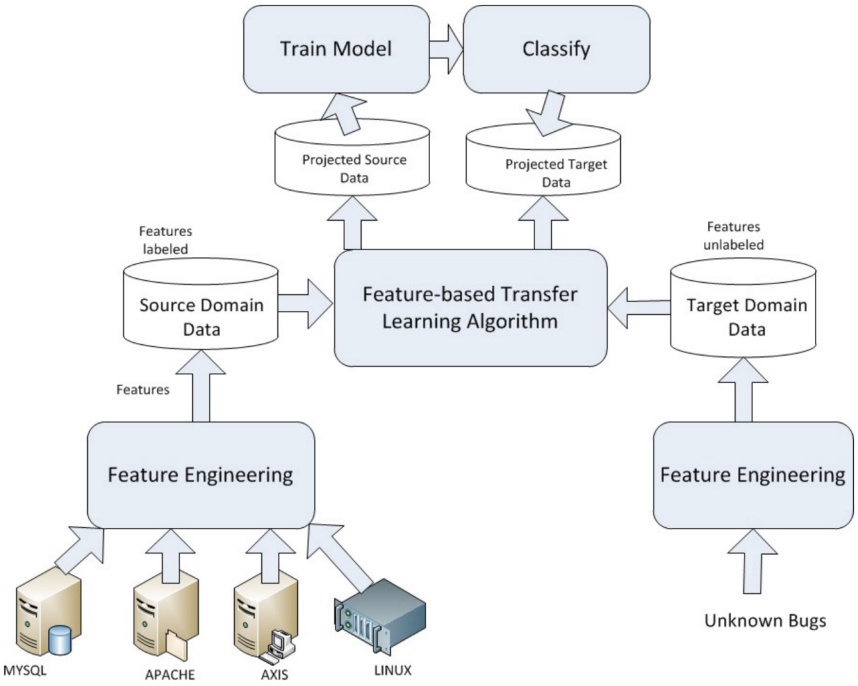

Fig. 2. Feature Engineering and Transfer Learning Framework

The source domain and target domain represent the different system environments but have same Mandelbug types with heterogeneous features. Here heterogeneous features means that the source and target domains are having different features but same number of observations unlike binary classification. For example, the source domain includes Apache and Linux kernel bugs related to TIM category while target domain includes Axis bugs related to TIM subcategory. So, the FETLA framework allows us to add new features to the target domain and this is absolutely possible in any critical operational environment where sophisticated system operations may change due to evolving Mandelbugs, which results in identifying new features. This justify the different Mandelbug feature distributions in the source and target domains.

Our FETLA framework operates on three core phases: (1) Raw text data processing, (2) Feature engineering, and (3) Feature-based Transfer Learning. First, feature sets are extracted from the raw text data. Second, a good working heterogeneous feature space dataset is generated through feature-engineering process. Third, through TL method a new common latent feature space is learned from the source and target domain datasets [21]. The newly learned feature representations are then classified using baseline classifiers such as decision trees, SVM, Naive Bayes, KNN and random forest trees. These classifiers are trained using known samples of one category and then try to predict the newly learned samples from another different category.

4 Feature Engineering

In our work, the bug reports include technical words making the feature construction process complex to work with because available dictionaries don't include technical words. For example,"httpd","SIGKILL","free-proc chain","mod CGI","SSL", etc. So, we have extracted technical words by processing through dictionary and selecting non-dictionary words as technical words.

4.1 Advanced Text Processing

Preprocessing of the textual-description of bugs are done in four steps: tokenization, stemming, typos, and strikeouts.

Tokenization. First weak correlations are reomoved from the bug reports by performing removal of symbols, punctuation marks and numbers. Then tokens are produced by breaking text streams into meaningful phrases or words.

Stemming. Second, a well known stemming tool stemmer7 is used to remove derived words and replaced with the base word. For example, the words "driving", "driver", and "drived" all would be replaced with base word "drive".

Typos. Third, we have used NLTK [22] and python to correct any typos.

Strikeouts. This step is done during the collection of textual data. We wrote a program to detect strikeouts on html pages and remove it completely.

4.2 Feature Extraction

The bug report dataset have fields such as "Summary", "Description", and "Comments" all are in text based. Thus, in our feature extraction process, word tokens are extracted as features from the summary, description, and comments fields of a bug report. The extracted features are represented in terms of numerical values which are truly the frequency of word tokens in the bug report.

4.3 Tf-Idf

Term frequency - inverse document frequency highlights importance of frequently occured words in a given record, while reducing importance of frequently occured words in many documents at the same time.

5 Classifier Engine

5.1 Baseline Classifier

Now the bug dataset has the features such as bud-ID, Label, tf-idf, bug status, product name that produced the bug, hardware, component, and importance of that bug. We need to perform data preprocessing of the dataset to fit the baseline classifiers. We then follow the steps below to make our text dataset is appropriately converted to numericals:

- Apply label encoder to each column such as Status, Product, Component, and Hardware in our dataset – label encoder object knows how to understand word labels.
- Assign values to column "Importance" based on the importance of the bug. So, we assign higher numerical values to bugs with higher importance. E.g. P3 has higher importance than P2 and then P1. Table 1 shows a snapshot of assigned values.

Next, we performed baseline classification on the dataset and the results of baseline classification are shown in Table 2. Baseline classifiers didn't capture the mandelbug data properly as we can see it from the classification accuracy and F1-score. This can be multiple reasons for low accuracy and F1-scores such as discarding multiple useful features from the raw text, improper label encoding from text to numbers etc. As we don't have control over label encoding methods, an alternative solution is to include all the raw text data as features with different feature space. One such solution is the spectral transformation where all features are projected to a new common latent subspace with similar distribution of mandelbugs.

Table 1. Assigning values to feature named Importance

P3 Normal = 3.0	P2 Normal = 2.0	P1 Normal = 1.0
P3 Blocking = 3.05	P2 Blocking = 2.05	P1 Blocking = 1.05
P3 low = 3.1	P2 low = 2.1	P1 low = 1.1
P3 High = 3.2	P2 High = 2.2	P1 High = 1.2
P3 Major = 3.3	P2 Major = 2.3	P1 Major = 1.3
P3 Critical = 3.4	P2.Critical = 2.4	P1.Critical = 1.4
P3 Normal = 3.0	P2 Normal = 2.0	P1 Normal = 1.0
P3 Blocking = 3.05	P2 Blocking = 2.05	P1 Blocking = 1.05

Table 2. Baseline classification results

Model Name	Accuracy	F1-Score
Decision Tree	0.39	0.39
SVM Norm	0.39	0.39
Naive Bayes	0.23	0.23
KNN	0.31	0.31
Random Forest	0.36	0.36

5.2 Problem Formulation

Mandelbug classification is modeled as a multi-class classification problem, assigning a bug to a subcategory of Mandelbug type. The task here is to distinguish a bug subtype

from another. Thus, our multi-class classification problem is divided into multiple binary classification problem such as One-vs-One (OvO) heuristic approach. For example, our multi-class classification problem with four classes:'TIM','ENV','LAG', and'SEQ' are divided into six binary classification problems by OvO approach as follows:

1: TIM vs. ENV
2: TIM vs. LAG
3: TIM vs. SEQ
4: ENV vs. LAG
5: ENV vs. SEQ
6: LAG vs. SEQ

For the test dataset, a probability score for each class is calculated from each binary classifier model and the final class label is derived from the argmax of the sum of the scores (class with the largest sum score).

5.3 Transfer Learning Method vIA Spectral Transformation

Each binary classification problem can be learned through transfer learning method where one class is considered as source and other class as target. Let's assign our training examples to source domain training $A = \overrightarrow{x_i}, \overrightarrow{x} \in R^m$ with class labels $L_A = y_i$, and target domain data $B = \overrightarrow{u}, \overrightarrow{u} \in R^n$. The vector $\overrightarrow{x}$ is the extracted features from source domain data and $\overrightarrow{u}$ is the extracted features from target domain data. Both $\overrightarrow{x}$, and $\overrightarrow{u}$ have different distributions $P_A(x) \neq P_B(x)$, and different dimensions $R^m \neq R^n$. The main objective of our framework is to classify class labels of target domain (B) accurately.

In our work, datasets A and B corresponds to source and target data respectively. Next, A and B are projected into new k-dimensional latent subspace using spectral transfomation, where instances from one category are homogeneous and instances from other categories are discriminative. The new optimal projections of A and B are V_A, $V_A \in R^k$ and V_B, $V_B \in R^k$ respectively. Now, the objective is to optimize the V_A and V_B as follows:

$$\min_{V_A, V_B} \psi(V_A, A) + \psi(V_B, B) + \beta.\Delta(V_A, V_B) \tag{1}$$

where $\psi(*, *)$ is a distortion function that estimates the difference between the original data and projected data. $\Delta(V_A, V_B)$ is the difference between the projected datasets V_A, and V_B. β is a similarity contolling-parameter between the datasets V_A, and V_B.

We rewrite $\psi(V_A, A)$, $\psi(V_B, B)$ as follows:

$$\psi(V_A, A) = \|A - V_A P_A\|^2, \psi(V_B, B) = \|B - V_B P_B\|^2 \tag{2}$$

where V_A, $and V_B$ are estimated by a linear transformation. $\|*\|^2$ is the Euclidian norm of a matrix or matrix trace norm. The projected datasets A and B into a k-dimensional space denoted as ${P_A}^T \in R^{m \times k}$ and ${P_B}^T \in R^{n \times k}$ respectively. Indeed, the source and target can be represented as linear mapping matrices $P_A \in R^{k \times m}$ and $P_B \in R^{k \times n}$ respectively.

Next, We represent $\Delta(V_A, V_B)$ in terms of $\psi(*, *)$ as:

$$\Delta(V_A, V_B) = \psi(V_A, V_B) \tag{3}$$

Rewritting (1) by substituting (2) and (3) into (1) i.e. minimizing w.r.t $V_A, V_B, P_A, andP_B$, we obtain:

$$\min_{V_A^T V_A = I, V_B{}^T V_B = I} G(V_A, V_B, P_A, P_B) = \min_{V_A^T V_A = I, V_B^T V_B = I} \|\mathrm{A} - V_A P_A\|^2 + \|B - V_B P_B\|^2 + \beta.\left(\|V_B - V_A\|^2\right) \quad (4)$$

As (4) is not a convex problem, global minima can be obtained using gradient descent method. Taking the derivative w.r.t V_B (i.e. solving V_B, fix V_A, P_B, P_A):

$$\frac{\partial \mathrm{G}}{\partial V_B} = 2\left(V_B P_B P_B^T - B P_B^T + \beta(V_B - V_A)\right) \quad (5)$$

Similarly, solving V_A, fix V_B, P_B, P_A; solving P_B, fix $V_A, V_B, andP_A$; solving P_A, fix $V_B, V_A, andP_B$; we obtain

$$\frac{\partial \mathrm{G}}{\partial V_A} = 2\left(V_A P_A P_A^T - A P_A^T + \beta(V_A - V_B)\right) \quad (6)$$

$$\frac{\partial \mathrm{G}}{\partial P_B} = \left(V_B{}^T V_B\right)^{-1} V_B^T \mathrm{B} \quad (7)$$

$$\frac{\partial \mathrm{G}}{\partial \mathrm{P_A}} = \left(\mathrm{V_A}{}^{\mathrm{T}} \mathrm{V_A}\right)^{-1} \mathrm{V_A^T A} \quad (8)$$

6 Results

Here, we represented the evaluation of our proposed method using benchmark Mandelbug datasets.

6.1 Dataset

We used open-source software project hosted at tera-Promise data portal as benchmark dataset [7] and the dataset is in ".arff" format. The dataset contains bug reports from 4 projects such as OS-kernel (Linux), RDBMS (MySQL), HTTPD web server (Apache), and AXIS WS.

Our data columns represents the bug ID, bug type and bug subcategory type. We then run a script to filter out Mandelbug only (e.g. 2123, NAM, NAU). Further, we used the bug IDs on the on-line bug repositories of the open-source projects(for example, by putting the ID in the search field)to get more details about that bug. The repositories are available on the following urls: https://bugzilla.kernel.org/(Linuxkernel), https://bugs.mysql.com/search.php(MySQL), https://bz.apache.org/bugzilla/(ApacheHTTPD), https://issues.apache.org/jira/secure/Dashboard.jspa (Apache AXIS).

6.2 Experimental Settings

Mandelbug prediction is modeled as a binary classification problem to distinguish one Mandelbug type from another type To evaluate our approach, we arrange the source and target domain datasets in different settings such as TIM → SEQ, LAG → ENV, etc., where the arrow head points to the target domain data. Transfer learning approach allows to classifying the Mandelbugs with different feature spaces but same number of observations.

Transfer learning via spectral transformation approach and the baseline classifiers are implemented using python programming language (python version 3).

6.3 Evaluation

Our approach is evaluated by comparing the accuracy and F1-Score metrices of baseline classifiers with transfer learning approach. The baseline classifiers used in our work are Support Vector Machine (SVM), Decision Tree, Naive Bayes, KNN and Random Forests. Using results from Tables 3 and 4, one can observe that the using TL approach has significant improvement in accuracy (30% - 65%) and F1-score when compared with that of baseline classifiers.

Table 3. Accuracy of Mandelbug

Accuracy					
Datasets	DT	SVM	NB	KNN	RF
TIM → ENV	0.57	0.64	0.64	0.85	0.57
TIM → LAG	0.85	0.89	0.89	0.89	0.89
TIM → SEQ	0.91	0.91	0.91	0.91	0.91
LAG → ENV	0.90	0.85	0.85	0.85	0.82
LAG → ENV	0.8	0.8	0.8	0.8	0.8

Table 4. F1-Score of Mandelbug

F1-Score					
Datasets	DT	SVM	NB	KNN	RF
TIM → ENV	0.7	0.73	0.71	0.87	0.7
TIM → LAG	0.33	0.29	0.53	0.29	0.29
TIM → SEQ	0.16	0.0	0.0	0.0	0.0
LAG → ENV	0.93	0.9	0.9	0.9	0.88
LAG → ENV	0.89	0.89	0.89	0.89	0.89

7 Conclusion

Our work has developed and implemented a framework using feature engineering, NLP and transfer learning to categorize types of mandelbugs. Performance evaluation of our transfer learning approach in comparison with baseline classifiers shows that our approach has an enhanced accuracy of 30% to 65% in classifying variants of mandelbug compared to baseline classifiers. Also, it supports different feature spaces compared to baseline classifiers. In our future work, we'll investigate other available models to see how well the natural language data are captured and if possible to have improved classification accuracy. Furthermore, we'll also investigate all the possibilities of cyberattacks exploited by attackers in the presence of software bugs with a special case study of Mandelbug exploitation.

Acknowledgements. This work was partly supported by Targeted Infusion Project: Cybersecurity for Everybody - A Multi-Tier Approach to Cybersecurity Education, Training, and Awareness in the Undergraduate Curriculum by the National Science Foundation (NSF award #1912284).

References

1. Grottke, M., Nikora, A., Trivedi, K.: An empirical investigation of fault types in space mission system software (2010)
2. Grottke, M., Trıvedı, K.S.: Software faults, software aging and software rejuvenation special survey new development of software reliability engineering. J. Reliab. Eng. Assoc. Japan **27**(7), 425–438 (2005)
3. Grottke, M., Trivedi, K.: A classification of software faults. In: Supplemental Proc. Sixteenth International Symposium on Software Reliability Engineering, vol. 27, pp. 4.19–4.20 (2005)
4. Grottke, M., Trivedi, K.S.: Fighting bugs: remove, retry, replicate, and rejuvenate. Computer **40**, 107–109 (2007)
5. Cotroneo, D., Grottke, M., Natella, R., Pietrantuono, R., Trivedi, K.S.: Fault triggers in open-source software: an experience report. In: 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), pp. 178–187 (2013)
6. Grottke, M., Kim, D.S., Mansharamani, R., Nambiar, M., Natella, R., Trivedi, K.S.: Recovery from software failures caused by mandelbugs. IEEE Trans. Reliab. **65**, 70–87 (2016)
7. Cotroneo, D., Iannillo, A.K., Natella, R., Pietrantuono, R., Russo, S.: The software aging and rejuvenation, pp. 108–113 (2015) repository. http://openscience.us/repo/software-aging/
8. Gegick, M., Rotella, P., Xie, T.: Identifying security bug reports via text mining: An industrial case study. In: 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), pp. 11–20 (2010)
9. Arshad, F.A., Krause, R.J., Bagchi, S.: Characterizing configuration problems in java ee application servers: an empirical study with glassfish and jboss. In: 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), (Los Alamitos, CA, USA), pp. 198–207, IEEE Computer Society (2013)
10. Zaman, S., Adams, B., Hassan, A.E.: Security versus performance bugs: A case study on firefox. In: Proceedings of the 8th Working Conference on Mining Software Repositories, MSR '11, New York, NY, USA, pp. 93–102. Association for Computing Machinery (2011)
11. Xia, X., Zhou, X., Lo, D., Zhao, X.: An empirical study of bugs in software build systems. In: 2013 13th International Conference on Quality Software, pp. 200–203 (2013)

12. Thung, F., Lo, D., Jiang, L.: Automatic defect categorization. In: 2012 19th Working Conference on Reverse Engineering, pp. 205–214 (2012)
13. Xia, X., Lo, D., Qiu, W., Wang, X., Zhou, B.: Automated configuration bug report prediction using text mining. In: 2014 IEEE 38th Annual Computer Software and Applications Conference, pp. 107–116 (2014)
14. Wu, R., Zhang, H., Kim, S., Cheung, S.C.: Relink: Recovering links between bugs and changes. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE '11, (New York, NY, USA), pp. 15–25, Association for Computing Machinery (2011)
15. Sun, C., Lo, D., Wang, X., Jiang, J., Khoo, S.: A discriminative model approach for accurate duplicate bug report retrieval. In: 2010 ACM/IEEE 32nd International Conference on Software Engineering, vol. 1, pp. 45–54 (2010)
16. Sun, C., Lo, D., Khoo, S., Jiang, J.: Towards more accurate retrieval of duplicate bug reports. In: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), pp. 253–262 (2011)
17. Nguyen, A.T., Nguyen, T.T., Nguyen, T.N., Lo, D., Sun, C.: Duplicate bug report detection with a combination of information re- trieval and topic modeling. In: 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, pp. 70– 79 (2012)
18. Marcus, A., Maletic, J.I.: Recovering documentation-to-source-code traceability links using latent semantic indexing. In: Proceedings of the 25th International Conference on Software Engineering, ICSE '03, (USA), pp. 125–135, IEEE Computer Society (2003)
19. Haiduc, S., Aponte, J., Marcus, A.: Supporting program compre- hension with source code summarization. In: ACM/IEEE 32nd International Conference on Software Engineering, vol. 2, pp. 223–226 (2010)
20. Zhou, J., Zhang, H., Lo, D.: Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In: 2012 34th International Conference on Software Engineering (ICSE), pp. 14–24 (2012)
21. Zhao, J., Shetty, S., Pan, J.W.: Feature-based transfer learning for network security. In: IEEE Military Communications Conference (MILCOM), pp. 17–22 (2017)
22. Natural Language Toolkit (NLTK). Available online: https://www.nltk.org/