An Optimization Technique for Unsupervised Automatic Extractive Bug Report Summarization

Ashima Kukkar and Rajni Mohana

Abstract Bug report summarization provides an outline of the present status of the bug to developers. The reason behind highlighting the solution of individual reported bug is to bring up the most appropriate solution and important data to resolve the bug. This technique basically limits the amount of time that the developer spent in a bug report maintenance activity. The previous researches show that till date the bug report summaries are not up to the developer expectations and they still have to study the whole bug report. So, in order to overcome this downside, bug report summarization method is proposed in light of collection of comments instead of single comment. The informative and phraseness feature are extracted from the bug reports to generate the all possible subsets of summary. These summary subsets are evaluated by Particle Swarm Optimization (PSO) to achieve the best subset. This approach is compared with the existing Bug Report Classifier (BRC) and Email Classifier (EC). For all approaches, the ROUGE score was calculated and compared with three human-generated summaries of 10 bug reports of Rastkar dataset. It was observed that the summary subset evaluated by PSO was more effective and generated less redundant, noise reduction summary and covered all the important points of bug reports due to its semantic base analysis.

Keywords Natural language processing · Feature weighting · Summarization Unsupervised · Bug report · Particle swarm optimization

A. Kukkar · R. Mohana (✉)
Department of Computer Science, Jaypee University of Information Technology, Waknaghat, India
e-mail: rajnivimalpaul@gmail.com

A. Kukkar
e-mail: ashi.chd92@gmail.com

S. Bhattacharyya et al. (eds.), *International Conference on Innovative Computing and Communications*, Lecture Notes in Networks and Systems 56,
https://doi.org/10.1007/978-981-13-2354-6_1

1 Introduction

Other profession individuals have a myth that the software development includes only programming element; in contrast, this line of work has strong element of system information management. The software organization requires the management and formulation of artifacts like designs, source code with documentation, specifications, and bug reports. These bug reports are stored into the software bug repositories. Software bug repositories have extreme information to perform the work on project. Tester's needs to understand the artifacts related to project for recording the issues mentioned by customers and tracks the resolution of bugs. To find the main issue of a bug, a meaningful and huge conversation could happen between developer and reporter by bug reports. So the bug reports have a large amount of reported conversation in the form of messages from multiple peoples and these messages might contain few lines or multiple passages of unstructured text. For example, Mozilla bug #564,243 has 237 sentences and Mozilla bug #491,925 has 91 sentences as comments. To resolve the bug, the developer has to analyze all the comments in bug reports. This process can be tedious, time-consuming, burden, and very frustrating for the tester or developer. Sometimes, the amount of information might be overwhelming and sometimes, this information leads to duplicate, deserted, and non-optimized searches, just because the previous bug reports of the project has been ignored. In [1], the researcher suggested the way to reduce the developer effort and process time, consumed to identify the factual bug report is to provide bug summary. The ideal process of bug summarization is to develop a manually abstract of resolved bug by assigned developer. This summary is used by other developers for better understanding of the bugs. However, the ideal method is very difficult to use in practice because it demands excessive human effort. Therefore, it is need of automatic bug report summarization. The factual summary will save the developer efforts and time and helps in generating up to date summaries of the project on demand. This paper is focused on four challenges. The first challenge is extractive bug report summarization that has huge amount of information as comments and large search space cause NP-complete problem. When working with extensive comments containing a lot of words, sentence selection and sentence scoring become a difficult job and also affect the accuracy and speed of summarization. The second challenge is to increase the ROUGE score by selecting effective semantic text. For example, if there are 20 text lines to summarize and the user wants 20% summarization, then a total of 2^5 subsets are possible of semantic structure; if checked one by one, then this problem goes to intractable. The third challenge is sparsity of data. For example, suppose that one query term is not in the document, then the probability of occurrence of that term is zero to the document. If the probability of query term would be multiplied with the probability of every term of document, then the document would not be retrieved. The fourth challenge is reduction of information. Some high important features are assigned higher weight and low important features are assigned low weight. Feature Selection (FS) provides various heuristics methods which led to non-exhaustive search and reduced high-dimensional space by selecting the features. This drops the information and decreases the accuracy of summary by only selecting the higher weight features.

In such situations, the adoption of meta-heuristic, n-gram, and feature weighting (FW) methods is beneficial in accomplishing the optimal solution to resolve the problem in a productive way. Previously, there are two methods used for textual bug report summarization. First one is supervised learning-based method used by various researchers in [2–5]. The bottom line of this method is as follows: initially bug reports are manually summarized as training set, then text features are extracted from summarized bug reports, and statistical model are trained with these reports. Thereafter, this model is used to determine the features; further, these features are used to predict the manual summaries presented in training set. Later, the features are extracted from new bug report and its summary is predicted by using trained model. Second method is unsupervised learning-based method presented in [6–9]. The basic process of this method is as follows: In this, the centrality and diversity of the sentences are measured in a bug report to select the relevant sentences to put into the summary. In order to create effective, flexible bug report summary, reduce the time, effort of developer, and resolve above four challenges, we proposed an intelligent method for searching the effective semantic text by particle swarm optimization approach with concatenate scores of informativeness and phraseness methods for significant improvement in extractive bug report summarization.

Organization: This paper is divided into five sections: Sect. 2 presents an analysis by producing the summary of existing work related to this paper, Sect. 3 demonstrates the proposed system and algorithm in detail, and Sect. 4 explains the experiments and results thus exhibited along with an analysis of performance parameters. At last, the paper concludes and proposes the future work to be done in Sect. 5.

2 Related Work

Bug report summarization is the well-known research area in software industry, through the summarization dates back to 1958 when Luhn [10] generated the literature abstract. Text summarization process has been practiced in various domains like social media [11], videos [12], news articles [13], and audio [14]. From the past years, summarization process was also used by the incorporated world like Microsoft's Office Suite [15], Bug Triage Process [16], and IBM's Intelligent Miner for Text [17].

With the time, the summarization method is improved from simple term frequency methods to complex machine learning and natural language processing techniques.

Automatic bug report summarization is very difficult job; the vital challenge is how the sentences are picked to generate bug reports summary. To overcome this problem, the summarization methods are classified into supervised and unsupervised learning methods to select the appropriate sentences.

2.1 Supervised Learning Methods

In 2010, Rastkar et al. [1] used existing classifier like Email and Meeting Classifier (EMC), Bug Report Classifier (BRC), and Email Classifier (EC) to generate extractive summary. The human annotators generated the summary of 36 bug reports from (Mozilla, KDE, Eclipse, Gnome), and the bug report classifier performed better than other two classifiers by having <62% precision. Further in 2014, Murphy et al. extended the version of [18] by using the same classifier, to check whether the generated summaries helped in detecting the duplicates bugs. They performed task-based evaluation and found that summaries save the time as well as helped the developers in detecting the duplicates bug without the evidence of decreasing accuracy.

2.2 Unsupervised Learning Methods

The work by Mani et al. in 2012 [19] used four unsupervised learning methods to create the bug report summary. At first, noise reducer is developed to find out the inessential sentences from the reported bug reports after that Diverse Rank, Centroid, Grasshopper, and Maximum Marginal Relevance (MMR) approaches were applied on IBM DB2 bug report. These methods choose the sentences which were central to the bug reports. The author also compared this unsupervised summarization method to supervised method of [1] and found better results. Another work in 2012 by Lotufo et al. [20] modeled user reading process by three hypothetical model and applied heuristics using Markov chain and PageRank method to rank the sentences according to the probability of reading and generate summaries by choosing higher probability sentences. The precision of the model is improved up to 12% as compared to EC in [1] on Mozilla, Launchpad, Chrome and Debian projects. In 2016, Ferreira et al. [21] used Euclidean Distance (ED), Cosine Similarity (CS), PageRank, and Louvain community detection (LCD) algorithms to generate the summary from the comments instead of isolated sentences. It was observed that the ranking of the most appropriate sentences helped the developers in finding the relevant information as compared to manually generated summary of Angular, Bootstrap, and jQuery projects.

3 Proposed Summarization System

This section presents the proposed approach and proposed algorithm to generate the bug report summary.

3.1 Proposed Approach

The overall proposed system process is shown in Fig. 1; further, this is dived into summarization process by using particle swarm optimization. At first, the bug reports (free text) are taken as input, and this text is tokenized. It is then disintegrated into the sets of paragraphs, D = {PP1, PP2, PP3, …}. Every paragraph is break down into the number of sentences. P = {SS1, SS2, SS3, …}, each sentence contains many terms (features), S = {FF1, FF2, FF3, …}. The output of tokenization task is segregated words, paragraph, and sentences. It has three main tasks: stop word removal, rooting extraction, and synonym replacement. The output of this phase is normalized text. The weights of all terms are concatenated to calculate the score of the terms in our system to provide the effective summary. These terms scores are added to sentences to gain the sentence score. After preprocessing, the two features scoring methods are used based on informativeness and phraseness principles [22]: TF-IDF and N-gram method. The bigram, trigram features are extracted using Eqs. 4–7 to preserve the semantics to bug summary, which reduced the sparsity of the text [23]. The TF-IDF features are extracted using Eqs. 1–3 from the normalized data to reflect how much the term provided the information about a bug report [21]. The matrix is considered for every bigram and trigram term. The number of rows and column showed the n-gram words (W) number that are extracted from these bug reports and their probabilities (P). After all the terms are extracted from bug reports dataset, the probability of each term is calculated by Eq. 8. This probability score and TF-IDF score are concatenated to get the combined score of each term. Instead of feature selection (FS), the feature weighting (FW) method is adopted. FW is a generalization from of FS which involves much larger searching space and have more flexibility to assign the continuous relative weight over FS. So these term weights (score) are added to the sentences to gain the sentence score, and the subsets of summary are produced by sentences according to user input, and subset score is calculated for each subset. The subsets of summary are chosen by using Particle Swarm Optimization (PSO) algorithm [24] due to its selecting nature of optimizing searching. It has global and local optimization to find effective semantic text and sentence, which increased the rogue score. This subset act as particles and subset score act as initial position.

Suppose S_i is the produced summary according to user input and has n subsets, $S = \{s_1, s_2, s_3, \ldots, s_n\}$. These sentences have corresponding weights W_k, $W = \{W_1, W_2, W_3, \ldots, W_n\}$. By assigning weight to each sentence, WS is produced, $WS = \{W_1S_1, W_2S_2, W_3S_3, \ldots, W_nS_n\}$. PSO algorithm is described as follows: At first, initialize the each particle's position to subset score. The swarm is updated at every cycle with the best value based on Eqs. 9–10. The position best (pb_i) is taken as the fitness value of summary and the global best (gb) is taken as the fitness value found by swarm best of personal bests. The new fitness value is compared to previous value of pb_i. If new fitness value is surpassed, then the previous value of pb_i is updated to new pb_i value. If the pbi value has not changed for any particle, then new pb_i is compared with previous gb value. If the new pb_i is surpassed than the previous gb, the value of pb_i is selected as new gb. At each iteration, the gb presented the position

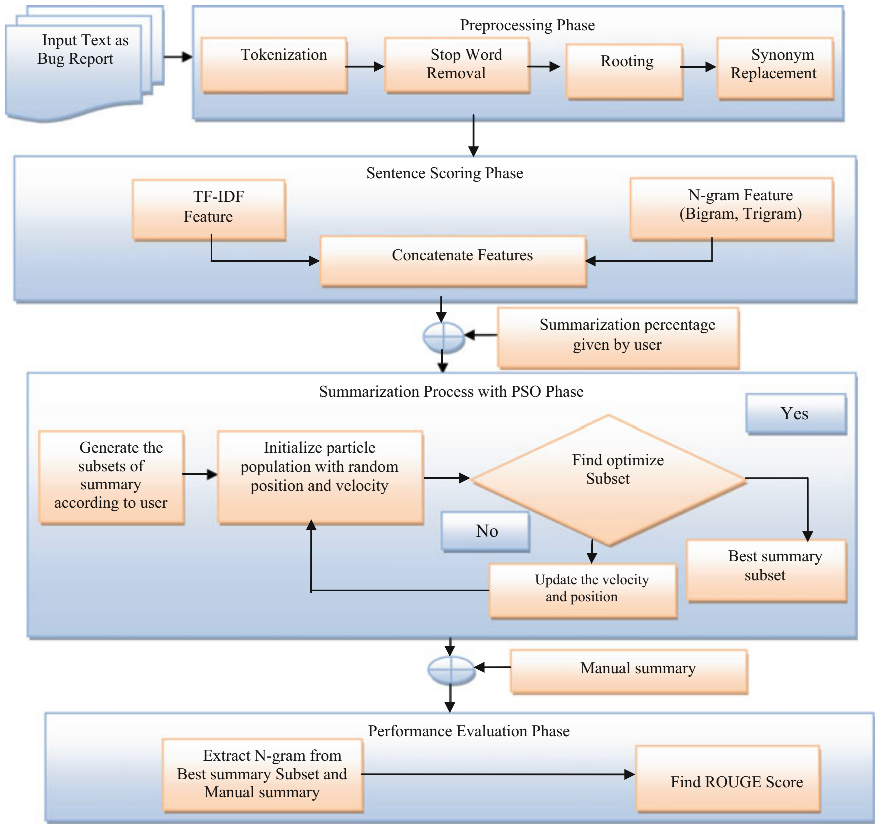
Input Text as Bug Report
Preprocessing Phase
Tokenization
Stop Word Removal
Rooting
Synonym Replacement
Sentence Scoring Phase
TF-IDF Feature
N-gram Feature (Bigram, Trigram)
Concatenate Features
Summarization percentage given by user
Summarization Process with PSO Phase
Yes
Generate the subsets of summary according to user
Initialize particle population with random position and velocity
Find optimize Subset
No
Update the velocity and position
Best summary subset
Manual summary
Performance Evaluation Phase
Extract N-gram from Best summary Subset and Manual summary
Find ROUGE Score

Fig. 1 Overall proposed system for bug report summarization

of the particle and showed a vector for best selected subset of the current bug report. The optimized summary is found after several iterations until the condition is not met. At last, ROUGE score [25] is calculated using Eq. 11.

3.2 Proposed Algorithm

```
Algorithm1: The summarization Process with PSO
Input: The XML files of bug reports
Output: ROUGE Score of optimized summary of bug reports
Step 1: Input text for preprocessing.
Step 2: Extract the TF-IDF Features
    for ( i = 1; i <=1;i++){
        $TF(t,D) = \frac{f_D(t)}{\min_{\omega \varepsilon D} f_D(\omega)}$    (1)
        $IDF(t,D) = ln\left(\frac{|d|}{|\{D \epsilon D : t \epsilon D\}|}\right)$    (2)
        $TFIDF(t,D,d) = tf(t,D) \cdot idf(t,d)$    (3)
$F_D(t) :=$ frequency of term t in document D
$d :=$ Documment Corpus
    }
i is number of feature.
Step 3: Extract the Bigram and Trigram Features
For bigram
        $P(Z_n|Z_0 \dots\dots\dots Z_{n-1}) \approx P(Z_n|Z_{n-1})$    (4)
        $P(Z_1^k) = \prod_{n=1}^{k} P(Z_n|Z_{n-1})$    (5)
For trigram
        $P(Z_n|Z_0 \dots\dots\dots Z_{n-1}) \approx P(Z_n|Z_{n-2}Z_{n-1})$    (6)
        $P(Z_1^k) = \prod_{n=1}^{k} P(Z_n|Z_{n-2}Z_{n-1})$    (7)
$P(Z_n)$:= Probability of word nth Z in document d
$Z_{1,2,3\dots k}$= Word Sequence
Step 4: Concatenate both feature.
    $P_{total}(P_i) = TF - IDF(P_i) + bi - gram(P_i) + tri - gram(P_i)$    (8)
Step 5: Make the subset of Summary produced by features and user input.
        y= $(\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, \dots\dots\dots\dots\dots\dots.. \boldsymbol{2^n})$
y:= is feature subset.
Step 6: Input these subset y in optimizer PSO as the particles form. Until an optimized subset
is met, repeat the following:
Step 7: In PSO model for each particle j in S sentence , do for each dimension d in D
do
//Initialize each particle's position and velocity to random values.
        $X_{j,d} = Rnd(x_{max}, x_{\min})$
        $sv_{j,d} = Rnd(-v_{max}/3, v_{max}/3)$
end for
//Initialize particle's best position and velocity
        $sv_j(l+1) = sv_j(l) + \gamma 1_i(pb_j - x_j(l)) + \gamma 2_i(gb - x_j(l))$    (9)
New velocity
        $x_j(l+1) = x_j(l) + sv_j(l+1)$    (10)
```

Where,

j	Particle Index
1	Discrete time Index
sv_j	Velocity of jth Particle
x_j	Current Position of jth Particle
pb_j	Personal Best Position of jth Particle
gb	Global Best Position
$\gamma 1_i$ and $\gamma 2_i$	Random number Between the [0,1]applied to the jth Particle

```
Step6: Set the global best position to its initial position
                    if$f(x_i) > f(pb_i)$
                        $pb_i = x_i$
// update global best position
                    if$f(pb_i) > f(gb)$
                        $gb = pb_i$
else
end if
end for
Step 7: Optimize subset is given by PSO, find ROUGE score.
Step 8:Optimizer summary and Manual Summary Concatenate for extraction of N-gram features
Step 9: Find the Rouge Score by the following formula
```

$$\frac{\text{No\_of\_Overlapping\_Words}}{total\_words\_in\_refernce\_summary} \quad (11)$$

4 Experimental Result

In this paper, we used the benchmark Rastkar dataset [1, 18]. The ROUGE score is calculating based on three manual summaries of each bug report. Based on the above approach, two experiments are conducted to analyze the approach using ROUGE evaluation metric at 30% summary percentage given by user. If we decrease the summary percentage, then the important information is lost. If we increase the summary percentage, then unwanted data is included in the summary. In this experiment, the bug report summary is generated of every single bug report by using PSO optimization technique. The basic aim of this experiment is to overcome the four challenges which are discussed in the introduction section. This generated summary is compared with three human-generated summary [1, 18], and ROUGE score is calculated as demonstrated in Table 1.

It can be seen from Table 1. The range of ROUGE score based on PSO-generated summary of 10 bug reports was from 72 to 97%. The range of ROUGE score based on BRC and EC-generated summary of 10 bug reports was from 30 to 93% and 51 to 92%. The summary subsets evaluated by PSO were more effective and generated less redundant summary and covered all the important points of bug reports because PSO used n-gram approach for matching which is by default a semantic relation.

Table 1 Comparison of ROUGE score with manual summary using PSO, bug report classifier, and email classifier

Bug report no.	ROUGE score using PSO			ROUGE score using bug report classifier [1, 18]			ROUGE score using email classifier [1, 18]		
	Rouge-1	Rouge-2	Rouge-3	Rouge-1	Rouge-2	Rouge-3	Rouge-1	Rouge-2	Rouge-3
Bug 1	79.14	75.83	75.17	92.03	70.75	33.88	79.88	52.067	51.2
Bug 2	88.54	82.81	80.42	88.36	85.25	34.38	86.47	57.65	55.98
Bug 3	9.33	92.12	89.29	84.64	81.65	33.24	90.86	59.83	56.29
Bug 4	91.84	88.96	80.96	74.53	68.12	30.96	86.46	57.38	53.78
Bug 5	86.60	85.34	82.59	75.34	73.23	36.63	84.80	55.57	54.19
Bug 6	86.23	83.82	82.61	70.56	60.78	45.67	87.46	58.28	55.57
Bug 7	96.65	93.07	86.13	86.28	83.26	35.64	91.58	59.88	56.41
Bug 8	95.65	89.57	86.09	91.87	88.32	34.88	89.19	58.51	56.28
Bug 9	90.34	88.73	84.62	92.27	90.14	35.46	89.51	59.15	57.17
Bug 10	96.00	92.00	88.00	90.93	86.65	34.77	91.81	59.94	57.75

5 Conclusion

It can be concluded that the bug report summarization is very important task for bug triage process. The developer or user reported the report in BST after observing the unexpected behavior of software. With the passage of time, valuable information is accumulated of observed issues. These bug reports are not easy to read. They contained the free text as comments, discussions, and opinions about the bug fixing and bug resolving. So the reading task of these bug reports is time-consuming and tedious. To save the developer time and effort by not reading the entire bug report, need of an Extractive summary generation model of entire bug report. A novel approach is proposed to generate the bug report summary based on collection of comments. The PSO optimization technique is used to evaluate the appropriate summary subset. These subsets are generated by using informative and phraseness feature extraction methods. This method is compared with bug report classifier and email classifier. The summary subset evaluated by PSO was more effective and generated less redundant summary and covered all the important points of bug reports due to its semantic base analysis. In future work, some other optimized techniques will be applied with it to enhance the accuracy of the summarization.

Acknowledgements We are mightily thankful to researchers Gail C. Murphy, Sarah Rastkar from the University of Columbia for proving the dataset. This helped us a lot in our research [1, 18].

References

1. Rastkar S, Murphy GC, Murray G (May 2010) Summarizing software artifacts: a case study of bug reports. In: Proceedings of the 32nd ACM/IEEE international conference on software engineering-volume 1. ACM, pp 505–514
2. Nomoto T, Matsumoto Y (July 2002) Supervised ranking in open-domain text summarization. In: Proceedings of the 40th annual meeting on association for computational linguistics. Association for Computational Linguistics, pp 465–472
3. Wong KF, Wu M, Li W (August 2008) Extractive summarization using supervised and semi-supervised learning. In: Proceedings of the 22nd international conference on computational linguistics-volume 1. Association for Computational Linguistics, pp 985–992
4. Yue Y, Joachims T (July 2008) Predicting diverse subsets using structural SVMs. In: Proceedings of the 25th international conference on machine learning. ACM, pp 1224–1231
5. Li L, Zhou K, Xue G.R, Zha H, Yu Y (April 2009) Enhancing diversity, coverage and balance for summarization through structure learning. In: Proceedings of the 18th international conference on World wide web. ACM, pp 71–80
6. Mihalcea R, Tarau P (2004) Textrank: bringing order into text. In: Proceedings of the 2004 conference on empirical methods in natural language processing
7. Erkan G, Radev DR (2004) Lexrank: Graph-based lexical centrality as salience in text summarization. J Artif Intell Res 22:457–479
8. Zhai C, Cohen WW, Lafferty J (June 2015) Beyond independent relevance: methods and evaluation metrics for subtopic retrieval. In: ACM SIGIR forum, vol 49, no 1. ACM, pp 2–9
9. Zhang B, Li H, Liu Y, Ji L, Xi W, Fan W, Ma WY (August 2005) Improving web search results using affinity graph. In: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, pp 504–511

10. Luhn HP (1958) The automatic creation of literature abstracts. IBM J Res Dev 2(2):159–165
11. Lin YR, Sundaram H, Kelliher A (April 2009) Summarization of large scale social network activity. In: IEEE international conference on acoustics, speech and signal processing, 2009 ICASSP 2009. IEEE, pp 3481–3484
12. Han B, Hamm J, Sim J (January 2011) Personalized video summarization with human in the loop. In: IEEE workshop on applications of computer vision (WACV), 2011. IEEE, pp 51–57
13. Radev DR, Blair-Goldensohn S, Zhang Z, Raghavan RS (March 2001) Newsinessence: a system for domain-independent, real-time news clustering and multi-document summarization. In: Proceedings of the first international conference on Human language technology research. Association for Computational Linguistics, pp 1–4
14. Waibel A, Bett M, Metze F, Ries K, Schaaf T, Schultz T, Zechner K (2001) Advances in automatic meeting record creation and access. In: 2001 Proceedings of IEEE international conference on acoustics, speech, and signal processing, 2001 (ICASSP'01), vol 1. IEEE, pp 597–600
15. Microsoft office suite (2012). http://office.microsoft.com/en-us/word-help/automatically-summarize-a-document-HA010255206.aspx
16. Zhang T, Jiang H, Luo X, Chan AT (2016) A literature review of research in bug resolution: tasks, challenges and future directions. Comput J 59(5):741–773
17. Intelligent miner for text (2012). http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?Subtype = ca\&infotype = an\&appname = iSource\&supplier = 897\&letternum = ENUS298-447
18. Rastkar S, Murphy GC, Murray G (2014) Automatic summarization of bug reports. IEEE Trans Softw Eng 40(4):366–380
19. Mani S, Catherine R, Sinha VS, Dubey A (November 2012) Ausum: approach for unsupervised bug report summarization. In: Proceedings of the ACM SIGSOFT 20th international symposium on the foundations of software engineering. ACM, p 11
20. Lotufo R, Malik Z, Czarnecki K Modelling the 'hurried' bug report reading process to summarize bug reports. Empir. Softw. Eng. 20(2):516–548
21. Ferreira I, Cirilo E, Vieira V, Mourao F (September 2016) Bug report summarization: an evaluation of ranking techniques. In: 2016 X Brazilian symposium on software components, architectures and reuse (SBCARS). IEEE, pp 101–110 (2015)
22. Han EHS, Karypis G, Kumar V (April 2001) Text categorization using weight adjusted k-nearest neighbor classification. In: Pacific-Asia conference on knowledge discovery and data mining. Springer, Berlin, Heidelberg, pp 53–65
23. Verberne S, Sappelli M, Hiemstra D, Kraaij W (2016) Evaluation and analysis of term scoring methods for term extraction. Inf Retr J 19(5):510–545
24. Kennedy J (2011) Particle swarm optimization. In: Encyclopedia of machine learning. Springer US, pp 760–766
25. Lin CY (2004) Rouge: a package for automatic evaluation of summaries. Text Summ Branches Out