PAPER

Automated Duplicate Bug Report Detection Using Multi-Factor Analysis

Jie ZOU[†], Ling XU^{†,††a}, Mengning YANG^{†,††}, Xiaohong ZHANG^{†,††}, Jun ZENG[†], *Nonmembers*,
and Sachio HIROKAWA^{†††}, *Member*

SUMMARY The bug reports expressed in natural language text usually suffer from vast, ambiguous and poorly written, which causes the challenge to the duplicate bug reports detection. Current automatic duplicate bug reports detection techniques have mainly focused on textual information and ignored some useful factors. To improve the detection accuracy, in this paper, we propose a new approach calls LNG (LDA and N-gram) model which takes advantages of the topic model LDA and word-based model N-gram. The LNG considers multiple factors, including textual information, semantic correlation, word order, contextual connections, and categorical information, that potentially affect the detection accuracy. Besides, the N-gram adopted in our LNG model is improved by modifying the similarity algorithm. The experiment is conducted under more than 230,000 real bug reports of the Eclipse project. In the evaluation, we propose a new evaluation metric, namely exact-accuracy (EA) rate, which can be used to enhance the understanding of the performance of duplicates detection. The evaluation results show that all the recall rate, precision rate, and EA rate of the proposed method are higher than treating them separately. Also, the recall rate is improved by 2.96%-10.53% compared to the state-of-art approach DBTM.

key words: duplicate bug reports detection, topic model, LDA, N-gram, LNG

1. Introduction

With the increasing size of software projects, software is becoming more complicated. Maintenance costs are up to 2/3 of a software life cycle expense. Software bug reports are document descriptions about possible defects or errors found by software testers or users while the maintenance of software. Along with the increasing project scale and version evolution, various users submit bug reports to bug-tracking management systems. Thus, open source software like Eclipse, Firefox, and Open Office produce a great deal of duplicate bug reports. Taking Firefox as an example, the percentage of duplicate bug reports is up to 30% [1]. “Every day, almost 300 bugs appear that need triaging. This is far too much for only the Mozilla programmers to handle”, a programmer from Mozilla reported in 2005. Duplicate bug reports not only occupy hardware resources but also cause

problems with bug report assignment, increasing the already high cost of software maintenance.

In order to relieve the burden of manual detection of duplicate bug reports, there are many methods developed. Initially, there was an information retrieval (IR) method applied to this problem called the Vector Space Model (VSM) [2]. Then Natural Language Processing (NLP) was combined to improve the experimental results [3], [4]. After that machine learning (ML) was adopted as another predominant approach [5]–[7]. However, most of them ignored some important factors for duplicates detection, they have primarily focused on textual information and none of them take all five types of features into account: textual information, semantic correlation, word order, contextual connections, and categorical information.

In duplicate bug reports, the keywords are usually the same because of the implicit consistency, like “error”, “open”, “close”, and “inherited”, their textual similarity is very high. Also, the bug reports are usually written by different reporters with different personal habits, they would describe the same technical issue(s) with different terms, thus the duplicate bug reports might not be textually similar in this case. The correlation degree often depends on the semantic correlation, instead of repeated words and expressions. For example, “check up” and “look over” are not textually similar but semantically similar. The textual information and semantic correlation from the bug reports measure the relevance between two bug reports from the external performance and internal characteristics respectively. The semantic correlation makes up the defects of textual information in the parts not similar in the texts. It also helps reduce the uncertainty and imprecision of duplicate bug reports described in natural languages. However, the semantic measure is usually based on bag-of-words theory which ignores the word order in the bug reports, and since it is a way of reducing the dimension of the text content, its comparison of the same textual tokens is less powerful than textual similarity. Thus, they can complement each other, so using both of them can make it possible to consider both external and internal behaviors in duplicate bug report detection.

Moreover, a sentence in bug reports is not a random clustering of words. Instead, it follows language structure rules and specifies the sequence. While two documents may exhibit similar word frequencies, the issue described may not be similar and hence have a different context. For instance, we look at Eclipse bug ID 126039 and bug ID

Manuscript received February 1, 2016.

Manuscript publicized April 1, 2016.

[†]The authors are with the School of Software Engineering, Chongqing University, Chongqing 401331, P.R. China.

^{††}The authors are with the Key Laboratory of Dependable Service Computing in Cyber Physical Society Ministry of Education, Chongqing 400044, P.R. China.

^{†††}The author is with the Research Institute of Information Technology, Kyushu University, Fukuoka-shi, 812–8581 Japan.

a) E-mail: xuling@cqu.edu.cn

DOI: 10.1587/transinf.2016EDP7052

Fig. 1 The overall process of duplicate bug reports detection using LNG.

140192. When considering only the word frequencies, their similarity is 0.73, so they may be considered duplicates. On the other hand, when we take the word order factor into account, their similarity is only 0.28, and thus they would not be considered duplicates. The events of the bug occurrence recorded in the bug reports is also sequential, the order of words reflects that sequence. Furthermore, it is likely that word order can assist in topic inference [8]. Also, It has been proven that the contextual connections considering the association of the before words and the later words should not be ignored for duplicates detection in [9]. The categorical information from the REP model [10] is also important, which includes Product, Component, Version, Priority and Type. Furthermore, our statistical results show that the percentage of the same product and component in duplicate bug reports is very high, about 86.07% and 75.84% respectively.

In this paper, to overcome the above shortcomings, we improve the N-gram similarity algorithm and propose a novel method called the LNG model that couples the LDA model and word-level N-gram model. The topic model LDA [11], is based on semantic processing, it can find the semantic relations of duplicate bug reports. The N-gram model, is based on word processing and context-sensitive likelihood of occurrence, it ensures the strict comparison of the same textual tokens in the case of high textually similar. By combining them, the LNG model takes the advantages of both fields: textual and topic similarity measurement. Compared with other approaches, the combination model (LNG) considered more factors. These factors include textual information, semantic correlation, word order, contextual connections, and other structural information like categorical information. Moreover, we also propose a new evaluation method, EA rate, which reflects the rate of ex-

act accuracy besides precision and recall measures. EA rate measure helps us to ensure how correct the approach performs neither missing a duplicate bug report nor wrongly detecting any redundant bug report as duplicate.

The process of the LNG modeling is illustrated in Fig. 1. The bug repository is organized as a data structure - a list of buckets. The key of each bucket is a master bug report, and then there are the duplicate bug reports with this master bug report. When duplicate bug reports are detected, we only consider the master bug report. As an example, this figure implies a training sample containing only two topics k_1 and k_2 . In this paper, we divided our procedure into three phases.

First, we trained the topic model LDA using the training sample, which yielded the conditional distribution of the words within the topics $p(w|z)$ and conditional distribution of topics within documents $p(z|d)$ of each master bug report. We also got the term frequency of each master bug report with the N-gram modeling step. When there is a new bug report, we get its $p(z|d_{new})$ by the trained $p(w|z)$ in LDA modeling step, and then using the topic distribution $p(z|d)$, obtain the LDA similarity with each master bug report using the cosine similarity method. On the other hand, we get its term frequency through the N-gram model, and then obtain the N-gram similarity with each master bug report.

Next, we linear combine the two similarities and apply a machine learning technique to automatically adjust the weights.

Finally, the categorical information was utilized to fine tune the similarity. When the final holistic similarity is greater than the set threshold value, the bug reports will be recommended as duplicate results.

The main contributions of this paper are as follows:

- We detect duplicate bug reports by considering more different factors that potentially affect the detection accuracy. In particular, we consider the following factors: surface textual information, internal semantic correlation, word order, contextual connections, and categorical information.
- We improve the N-gram similarity algorithm and propose a new method called LNG model, which takes the strength of both semantic features from LDA topic model and textual features from word-level N-gram model.
- The experimental evaluation on more than 230,000 real bug reports from the Eclipse project show that our LNG model improves the recall rate of the state-of-art approach DBTM by 2.96%-10.53%. In addition, we propose a new measure metric (EA rate) aside from the precision and recall rates to enhance the understanding of an approach performance.

The remainder of this paper is organized as follows:

Chapter 2 discusses the related work. Chapter 3 presents our approach to detailed duplicate bug report detection. Chapter 4 shows and discusses the experimental results. Chapter 5 presents the validity threats of our work. Chapter 6 makes a conclusion.

2. Related Work

To detect the duplicate bug reports, an increasing number of experts currently involve themselves into this field. Most of the early works were statistical IR approaches. In 2005, Anviket al. [1] built a statistical model, using the cosine similarity and were able to detect 28% of all duplicate bug reports on the Firefox project. Hiew et al. [1] applied VSM to detect duplicate bug reports, which models a bug report as a vector. VSM [12] was adopted in previous ways of automatic detection of duplicate bug reports. However, the corpus to be processed usually reaches into 100,000 documents which are extremely massive. Moreover, the vector space has a high dimension, sparse data, and noise problems which lead to inefficient execution and a low recall rate as well as precision rate. Sureka and Jalote [13] used a character N-gram based model to detect duplicate bug reports, which applied N-gram at the character level. Its dataset was 200,000 bug reports from the open-source Eclipse project. The recall rate was 33.92% for the Top 50 results on 1100 randomly selected test cases and 61.94% for 2270 randomly selected test cases with a title to title similarity of more than a pre-defined threshold of 50. Sun et al. [14] proposed an REP model, an advanced IR approach, to calculate the similarity between two bug reports. In addition to textual content in summary and description of bug reports, other non-textual aspects including product, component, and version were also utilized for detection. They also extended BM25F [15], which is a document similarity formula built upon Tf-Idf.

Runeson P. et al. [3] applied Natural Language Processing (NLP) on the bug reports with 30% precision. Wang XY

et al. [4] added execution information on the basis of Runeson P.'s research, the recall rate was around 93%, and the precision rate 67%. The method improved the recall rate greatly, but the execution information is difficult to be obtained for specific bugs and often unavailable for typical bug reports.

The Machine Learning (ML) method is also a popular method on duplicate bug reports detection. Jalbert and Weimer [5] developed a system that automatically classifies duplicate bug reports. They used textual similarity, surface features, and graph clustering to predict duplicate status. And then Tian et al. [10] extended Jalbert and Weimer's work [5] by utilizing REP. Based on the research of Runeson P. et al., Sun et al. [6] mapped the vectored and identified bug reports to a discriminative model, and then trained an SVM classifier to detect duplicate bug reports. Compared with Runeson P.'s method, Sun's precision rate was about 20% higher.

All the above methods are primary word-based methods, they can diagnose the bug reports which are lexical similar. However, it is difficult in solving the problems of synonymy and polysemy in bug reports, so semantic information is necessary to help further improve the performance of word-based methods.

Recently, much attention has focused on comparing and combining two different classes of approaches. Kaushik and Tahvildari [16] investigated on the performance of the traditional VSM and the topic based models, compared them in an evaluation on the Eclipse and Firefox projects. They achieved a recall rate of 60% and 58% on Eclipse and Firefox, respectively. Zhou and Zhang [17] proposed BugSim, a method related to SVM and BM25Fext. Nguyen et al. [7] proposed DBTM with a combination of T-Model and BM25F, they considered IR-based features and topic-based features. Their evaluation showed that DBTM could improve the state-of-the-art approaches by up to 20% in accuracy on Eclipse, OpenOffice, and Mozilla, the highest is about 86%, which is higher than other duplicate bug reports detection approaches. These combined methods improved the detection on duplicate bug reports to a certain degree, but they also ignored some important factors for duplicates detection. Similarly, in this paper, we couple the word-based N-gram model and topic-based LDA model. However, differing from their research, we not only considered the textual similarity, semantic similarity but also take word order, contextual and categorical features into account, improving the detection accuracy.

Other studies on duplicate bug reports also should be noted. Zhang et al. [18] presented an survey on the existing work on bug-report analysis. Aggarwal et al. [19] proposed a simpler software-literature context method to detect duplicate bug reports. Thung et al. [20] have developed a tool called DupFinder that implements a unsupervised duplicate bug report detection technique. Alipour et al. [9] proved that the contextual approach should be considered. Lazar et al. [21] presented an improved method to detect duplicate bug reports based on textual similarity measures. Rastkar

et al. [22] indicated that generated bug report summaries could provide help for duplicates detection tasks. Banerjee et al. [23] utilized common sequence matching for duplicate report detection on Firefox and Eclipse, achieved a duplicate recall rate above 70% on Firefox. Amoui et al. [24] designed and developed a search-based duplicate bug detection framework on BlackBerry. Lerch and Mezini [25] proposed a method that only uses stack traces and their structure, which was available without a written bug report. Gopalan and Krishna [26] proposed a clustering based approach to detect duplicate bug reports on Mozilla, Eclipse and Open Office projects. Zhang et al. [27] developed a novel technique for duplicate bug report detection and classification of bug reports, which reduced the time and effort of triages. They used SVM to check whether it was a duplicate bug report, and then evaluated it on the Mozilla project. Other researchers studied bug reports based on different types, quality and severity [28]–[30].

3. The Proposed Method

Our proposed method consists of mainly two parts. First, we construct LDA topic model and calculate the LDA similarity. Next one is to linearly combine the N-gram similarity with a parameter setting by weight training, and then utilizing the categorial information to fine tune the similarity. If the similarity is greater than the set threshold value, they are considered duplicates.

3.1 Data Preprocessing

Since the original report file is organized in the form of an XML file containing a lot of redundant information, we used the java SAX parser to extract four kinds of key information including the short summary, detailed description, categorial and duplication information. The short summary and detailed description are included in the textual information. The categorial information contains four parts of bug reports: classification, product, component and version. The duplication information is used to validate the accuracy of the experimental results.

A bug report consists of multiple fields. The fields in different projects may vary to some extent, but in general they are similar. Table 1 lists the main fields of bug reports and an example of Eclipse bug report #9.

Since there are some invalid bug reports in the training sample space, we need to remove them. If the title and content of the original bug report are all null, which means that this report is non-existent, it should be eliminated. For example, bug report ID 511.

The data preprocessing is followed by extracting data includes data cleaning, word splitting, stemming, synonyms transforming, stop words filtering, and lower case transforming. Note that, the data cleaning process is different from common text processing, because most of the description of the bug has invalid information expressed in a fixed sentence pattern, such as:

Table 1 The structure and an example of bug reports.

Field	Description	Example
bug_id	Bug id	9
creation_ts	Creation time of bug	2001-10-10 21:34
short_desc	Summary: concise description of the issue	VCM Implementation - disallow root resource to be passed
long_desc	Description: detailed outline of the issue, such as what is the issue and how it happens	The implementation has to be changed because a root resource might be passed where an IResource is expected. See also 1G8435U. NOTES: KM (4/26/01 3:54:24 PM) This PR is very old. It should either be fixed, moved to future, or obsolete. Obsolete. PRODUCT VERSION: Eclipse Platform 0.020
classification	Which classification the issue is about	Eclipse
product	Which product the issue is about	Platform
component	Which component the issue is about	Team
version	The version of the product the issue is about	2.0
resolution	Resolution status, e.g., FIXED, WONTFIX, DUPLICATE	WONTFIX
priority	The priority of the report, e.g., P1, P2, P3	P3
reporter	The reporter of the issue	Jean-Michel Lemieux

1. Fixed in HEAD.

Available in builds > N20071108-0010. Available in builds > N20071108-0010.

2. *** Bug 208441 has been marked as a duplicate of this bug. ***

The sentence pattern 1 shows that the bug has been revised in previous versions, while pattern 2 points out which bugs are marked as duplicate bugs. The two patterns mentioned above have no connection with the bug itself. Since there is a lot of this kind of information in the original file and it produces a large number of duplicate invalid information which will eventually have a great influence on the experimental results, they are removed according to the fixed patterns.

After data cleaning, we split sentences and terms for the bug reports. Next, the stem word was extracted to remove the affixes and get the root, thereby removing the difference among original words, adjectives and verbs. Synonym transforming is to remove the effect of the same vocabulary with different expressions. For example, “Control+Tab” which is a shortcut key, there are Control-Tab, Ctrl+Tab, Ctrl-Tab, CtrlTab and other different ways of expression due to varied personal habits. And then filtering the stop words because there are some high frequency insignificant words in natural languages, such as “a”, “is”, “about” and so on, in which often results in meaningless and expensive processing. The final step is lower case transforming, we converts all the words to lowercase in the experiments. In this paper, the preprocessing was carried out

using GATE [31] and Lucene [32].

3.2 Topic Modeling with LDA

To construct the topic model, we used natural language processing toolkit MALLET [33] (Machine Learning for Language Toolkit). By MALLET, we converted all the texts of the training sample spaces to featured sequences, and set the number of topics according to the size of the sample space.

3.2.1 Constructing Topic Model

In topic model [34], [35], the topic is represented by the conditional probability distribution of words in the vocabulary. Topic models have been applied to various software engineering research questions, such as software evolution [36], [37], software defect prediction [38], and software change message classification in our previous work [39]. The main problem of topic model is how to extract the topic from the corpus and analyze it. If viewed from the generative model perspective, every word of an article is obtained through the process of “choosing a topic according to a certain probability, and then choosing a word from it according to a certain probability”.

There are two principal methods of training and reasoning topic models. One is PLSA [40] (Probabilistic Latent Semantic Analysis), the other is LDA (Latent Dirichlet Allocation). PLSA mainly uses EM [41] (expectation maximization) algorithm while LDA adopts Gibbs sampling method [42]. In the PLSA, document probability is associated with a particular document and lack of natural methods to deal with new documents. With increasing document amount, the parameters to be estimated also increase linearly, so it is not suitable for a massive data set. Compared with the PLSA model, the LDA model is a completely generative model which regards blend weights of topics as random variables of T dimension parameters rather than a set of individual parameters directly related to the training data. The proposed method was based on LDA for the corpus.

LDA [43] supposes that all documents have K topics. Assuming that the K dimensional vector α is the parameter of the prior data distribution of the topic, $K \times V$ matrix β is the parameter of the distribution of the words (V is the sum of words) in the topic, i.e. $\beta_{ij} = p(w_j|z_i)$ is the probability of the word w_j in the i_{th} topic. Thus, we generate a document topic distribution and N topics. The probability of the N words is expressed as:

$$p(\theta, z, w) = p(\theta|\alpha) \prod_{n=1}^N p(z_n|\theta)(w_n|z_n, \beta) \quad (1)$$

Where θ is the topic distribution vector of the document, z is the topic vector of N dimensions, and w is the vector composed of the N words. Since θ and z are latent variables which cannot be observed, they are eliminated through marginal distribution from the left:

$$p(w|\alpha, \beta) = \int p(\theta|\alpha) \prod_{n=1}^N p(z_n|\theta)(w_n|z_n, \beta) d\theta \quad (2)$$

To corpus D which has M documents,

$$p(D|\alpha, \beta) = \sum_{d=1 \dots M} p(w_d|\alpha, \beta) \quad (3)$$

Thus

$$p(D|\alpha, \beta) = \prod_{d=1}^M \int p(\theta_d|\alpha) \left(\prod_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn}|\theta_d) p(w_{dn}|z_{dn}, \beta) \right) d\theta_d \quad (4)$$

As shown above, the process of constructing the LDA model is a process of achieving maximized parameters α and β of $p(D|\alpha, \beta)$.

We obtain $p(w|z)$, α and β after building a topic model, and then deduce the topics of the new unlabeled texts by the trained topic model and predict its corresponding topic distribution which is also the conditional probability distribution of the document in the topic space.

$$p(\theta, z|w, \alpha, \beta) = \frac{p(\theta, z, w|\alpha, \beta)}{p(w|\alpha, \beta)} \quad (5)$$

3.2.2 Calculate LDA Similarity

We used the constructed topic model to get the document-topic matrix, converted the entire document-topic matrix into a vector space, and then calculated the similarity between the bug reports in the testing sample space.

There are a lot of vector space similarity measurement methods: Kullback-Leibler Divergence, cosine similarity [12], and so on. The experiments results show that the final results of the Kullback-Leibler Divergence and the cosine similarity were almost the same, which could be interchanged. The cosine similarity threshold of 0.9 roughly corresponds to the Kullback-Leibler Divergence threshold of 0.45 while the cosine similarity threshold of 0.85 roughly corresponds to the Kullback-Leibler Divergence threshold of 0.5 in the dataset. In this paper cosine similarity was adopted. To calculate the similarity of the testing sample document $D1$ and $D2$, the vector of document $D1$ in the i_{th} dimension topic space is d_{1i} , and that of document $D2$ is d_{2i} . The similarity calculation formula is as follows:

$$Sim(d_1, d_2) = \frac{\sum_i (d_{1i} * d_{2i})}{\sqrt{\sum_i d_{1i}^2 * d_{2i}^2}} \quad (6)$$

Note that, the cosine value ranges from 0 to 1. The bigger the angle is, the smaller the cosine value is. From the relationship between space and similarity, we can find that if the cosine value is bigger, the space similarity or semantic similarity between two documents is higher.

3.3 N-gram Modeling

The N-gram model [44] is a frequently-used model based

Algorithm 1 The Improved N-gram similarity algorithm**Estimate weights**(BR1, BR2, N)**Input:** the two bug reports prepared to calculate similarity: bug report BR1, bug report BR2 and the N-gram parameter: N.**Output:** the similarity between BR1 and BR2. (The range is [0, 1], the two bug reports are more similar when the similarity is higher)**method:**

```

get N-gram items of BR1(gramBR1);
get N-gram items of BR2(gramBR2);
get the total term frequency(every item of gramBR1 and gramBR2);
for i From 0 to gramBR1.length do
  for j From 0 to gramBR2.length do
    if gramBR1[i] is not equal to gramBR2[j] then
      continue;
    end if
    count1  $\leftarrow$  count1 + 1;
    break;
  end for
end for
for m From 0 to gramBR2.length do
  for n From 0 to gramBR1.length do
    if gramBR2[m] is not equal to gramBR1[n] then
      continue;
    end if
    count2  $\leftarrow$  count2 + 1;
    break;
  end for
end for
similarity = (count1+count2)/ total term frequency;
Return similarity

```

on word processing, it is also widely used in text classification and information retrieve [44], [45]. N-gram means that a certain sentence is a contiguous sequence of n words in a given text (also can be letters or phrases). N can take any value depend on the dataset. Generally, the experiment result is better when $N = 3, 4$, or 5 . Give an example of 3-gram sequence, when the sentence is “Duplication Detection for Bug Reports”, the sequence is {Duplication Detection for, Detection for Bug, for Bug Reports}.

The N-gram algorithm is shown in Algorithm 1. As shown, we firstly get the N-gram items of bug reports and the term frequency, and then count the *count1* and *count2* by comparing two bug reports; finally we get the N-gram similarity.

The N-gram method adopted in this paper is word-level N-gram and the similarity algorithm is improved from the original one [46]. In the original N-gram similarity algorithm, the similarity of two bug reports is asymmetric. For example, the items of bug report BR1 is {word₁, word₂, word₃, word₄}, the items of bug report BR2 is {word₁, word₂, word₃, word₃}, the items of bug report BR3 is also {word₁, word₂, word₃, word₃}. Intuitively, it is obvious that the similarity of BR2-BR1 is not 1 since their items are not all the same, and it is less than the similarity of BR2-BR3 because the items of BR2 and BR3 are more similar than BR2 and BR1. However, when calculating the similarity using the original N-gram similarity algorithm, the similarity of BR1-BR2 is 0.75(6/8), the similarity of BR2-BR1 is 1(8/8), and the similarity of BR2-BR3 is 1(8/8). That is, the similarity of BR2-BR1 and BR2-BR3 are equal, both are

Table 2 The percentage of the same product and component in duplicate bug reports.

Categorical information	Amount(percentage)
Product	23959(86.07%)
Component	21112(75.84%)

1. It is against the previous thought. Furthermore, the similarity of BR1-BR2 and BR2-BR1 is not same which means it is asymmetric. In our N-gram similarity algorithm, we compare the items of bug report BR1 across the bug report BR2 first, and then compare the items of BR2 across the BR1. The two-way traversal ensures that both the similarity of BR1-BR2 and BR2-BR1 are 0.875(7/8), the similarity of BR2-BR3 and BR3-BR2 are 1(8/8). That is, our N-gram similarity algorithm is symmetrical and it is also consistent with the previous experiential thought. Furthermore, our proposed word-level N-gram model improves the recall rate result on the Eclipse dataset by nearly 3.4% on average.

3.4 The LNG Model

The LDA and N-gram model have different advantages in detection of duplicate bug reports. The N-gram model is stricter in comparison, which is better in the detection of duplicate bug reports written with the same textual tokens. The topic model LDA can detect the topic similarity between two bug reports even when they are not similar textually. To the bug report, it can be divided into two parts: the bug description by natural language and the bug runtime information. To the bug description part, it is vast and ambiguous, the topic model LDA will be better, as to the runtime information part, it is usually textual similar, the word-based model N-gram will be better. By combining the two models, the LNG model takes both of the advantages of them, and takes the textual information, semantic correlation, word order, and contextual connections features into account. In addition, the LNG model also considers the categorial information. These non-textual features clearly help improve the performance of duplicate bug reports detection [14]. They reflect the relation of bug reports to a certain degree. Take product and component as an example, our statistical results show that the percentage is very high, about 86.07% of duplicate bug reports belong to the same product and 75.84% belong to the same component as seen in Table 2.

The overall procedure of the LNG model is shown in Algorithm 2. The LNG method consists of five steps. First, we construct the training and test sample spaces. Next, we use MALLET tool to construct the topic model LDA in the training sample space, apply the LDA model to infer the topic of the test sample space, and calculate the LDA similarity between the documents through the cosine similarity method. Then, we extract the N-gram sequence and calculate the N-gram similarity through the terms frequency statistics. Finally, we combine the LDA similarity and N-gram similarity to obtain the total similarity. In addition, we utilize the categorial information to fine tune the similarity. If one of the categorial information of two bug reports is

Algorithm 2 The LNG Model Algorithm**Estimate weights**($B, N, T, SInf$)**Input:** B : bug report collection in the dataset, N : the N-gram parameter, T : the LDA parameter topic number, $SInf$:categorical information of bug reports.**Output:** the similarity of every two bug reports(The range is [0, 1], the two bug reports is more similar when the similarity is higher)**method:**Build training sample space S_{train} Create testing sample space S_{test}

//training for LDA model

TrainLDA(S_{train})

//applying the trained LDA model

ApplyLDA(S_{test})**for all** bug report $B \in S_{test}$ **do**

Get the docs-topic matrix

end for

//calculate the LDA similarity between every two bug reports

list($LDASim(B_{test1}, B_{test2})$)= $LDASimilarityCalc(B_{test})$ **for all** bug report $B \in S_{test}$ **do**

Extract the N-gram items

end for

//calculate the N-gram similarity between every two bug reports

Get the total term frequency

Count the same items

list($NgramSim(B_{test1}, B_{test2})$)= $NgramSimilarityCalc(B_{test})$

//combined the LDA similarity and N-gram similarity

list($Sim(B_{test1}, B_{test2})$)= WeightSum(list($LDASim(B_{test1}, B_{test2})$),list($NgramSim(B_{test1}, B_{test2})$))

//utilizing the categorical information to fine tune the sim, the weight of

for all categorical information $SInf \in$ {classification, product, component, version} **do** **if** ($B_{test1}.SInf = B_{test2}.SInf$) **then** **if** ($Sim(B_{test1}, B_{test2}) \leq 0.975$) **then** $Sim(B_{test1}, B_{test2}) = Sim(B_{test1}, B_{test2}) + 0.025$ **else** $Sim(B_{test1}, B_{test2}) = 1$ **end if** **end if****end for**

Return the similarity of every two bug reports

equal, increasing the similarity by 0.025 when the similarity is less than or equal to 0.975. When it is more than 0.975, we set the similarity to 1. The algorithm returns the similarity of every two bug reports, if the similarity is greater than the threshold value, they are set as duplicates.

In the step of the combining similarity between the LDA and the N-gram, we use a machine learning technique called Ensemble Averaging[†] and a linear combination.

$$Similarity = \alpha_1 * LDA_similarity + \alpha_2 * Ngram_similarity \quad (7)$$

Where α_1 and α_2 are the parameters to control the significances of the models in detecting duplicate bug reports. α_1 is the weight of the LDA model, α_2 is the weight of the N-gram model. If α_1 is equal to 1, and α_2 is equal to 0, it is the LDA model. If α_1 is equal to 0, and α_2 is equal to 1, it is the N-gram model. So the LDA model and the N-gram model are the specializations of the LNG model respectively. Since the sum of α_1 and α_2 is equal to 1, we only have to learn the LDA weight α_1 , then α_2 is equal to $1 - \alpha_1$. α_1 is learned by a simple cross-validation and a searching

[†]http://en.wikipedia.org/wiki/Ensemble_averaging

Fig. 2 Ensemble weight training.

algorithm.

Figure 2 shows the α_1 training algorithm. Function MAP and parameters α_1 are initialized first. Then we calculate the LDA similarity $LDASim$ and N-gram similarity $NgramSim$ between test bug reports. The two similarities are combined into Sim with the formula (7). The similarity values Sim are used to get the duplicate results L_{pred} . We take L_{pred} to evaluate a goal function MAP , which is used to find the optimized value of α_1 . The α_1 increased with a step of 0.01 every time. When we obtain the highest value for MAP , The optimal α_1 value will be returned. The goal function MAP in our algorithm is the mean average precision as previously proposed [14].

$$MAP(L_{test}, L_{pred}) = \frac{1}{|L_{test}|} \sum_{i=1}^{|L_{test}|} \frac{1}{index_i} \quad (8)$$

Where L_{test} is the real duplicate links in the testing set; L_{pred} is the ranked list of predicted links; $index_i$ is the index where the true duplicate group is retrieved for the i -th query. MAP measures how well the algorithm ranks the true links, using it as a goal function to train. Then, using the trained weights α_1 and α_2 to calculate the combination of text and topic similarity Sim . After that, the categorical information is added to get the final similarity. The higher the final similarity is, the more likely the test bug reports are duplicates.

4. Experiments

4.1 Data Sets

In our experiment, we use the bug report dataset of the Eclipse^{††} project, which is one of the most popular open source projects and has millions of users worldwide^{†††}. The bug reports are stored and tracked by the Bugzilla tool^{††††}.

For the evaluation, we collected 213,000 bug reports generated from 2001.10 to 2007.12. After removing the invalid bug reports, there are 211,843 remained, among which, 27,838 are marked as “duplicate”. And then a small test sample is constructed to validate the experiment results like other papers [7], [14]. In order to simulate the real source data to the maximum extent, we should keep the ratio of the non-duplicate and duplicate bug reports in the test sample space at the same rate as the real source data. In

^{††}<http://www.eclipse.org/>

^{†††}<https://bugs.eclipse.org/bugs/>

^{††††}<http://www.numberof.net/number-of-eclipse-users/>

the real source data, the rate of duplicate is 27,838/211,843, about 13%, so we also select 13% as duplicates, which contained 200 duplicate bug reports. Thus, there are totally about 1538 bug reports in the test sample space, the number of non-duplicate bug reports is 1338 (1538-200). In the experiment of this paper, we first randomly sampled 200 duplicate bug reports, and then added the related bug reports which are duplicates to each-other with the 200 bug reports from the Eclipse official website. The remaining non-duplicate bug reports were also randomly selected from the non-duplicate dataset as disturbances. Then, the test sample space was constructed completely, and the remaining was selected as the training data.

4.2 Evaluation Criteria

Precision and recall, which are commonly used to measure the accuracy of classification algorithms, are used to evaluate the effectiveness of LNG and other comparative methods.

$$RecallRate = \frac{N_{detected}}{N_{total}} \quad (9)$$

$$PrecisionRate = \frac{N_{detected}}{N_{detectedall}} \quad (10)$$

Where $N_{detected}$ is the number of correct experimental detections of duplicate bug reports, N_{total} is the total number of the actual duplicate reports, $N_{detectedall}$ is the total number of reports labeled as duplicate in the experimental detections (including the right and wrong). Both of the two metrics can evaluate the effectiveness of the results, but we wonder how exact correct the approach performs in some situation, the exact correct means neither missing a duplicate bug report nor wrongly detecting any redundant bug report as duplicate, when there is one duplicates, they can detect one duplicate, when there are two duplicates, they can detect two duplicates etc. So we propose a new measure metric aside from the precision and recall rates to enhance the understanding of an approach performance which is called EA rate.

$$EARate = \frac{N_{accuracy}}{N_{all}} \quad (11)$$

Where $N_{accuracy}$ is the number of bug reports whose duplicates are detected exactly correct, N_{all} is the total number of bug reports which is the real duplicate links in the testing sample space.

A high EA rate implies a high recall rate and a high precision rate, nor do high recall rate or high precision rate means high EA rate. In addition, the experiment to support the EA rate is shown in Fig. 4, Fig. 5, and Fig. 6 of Sect. 4.4.

4.3 Research Questions

We focus on the following four research questions:

RQ1) *How effective is our proposed word-level N-gram?* To the problem of whether our proposed word-level N-gram is

Fig. 3 Comparison with Sureka's N-gram.

working better or not, we compare the recall results of our improved N-gram similarity algorithm with Sureka's character N-gram to validate its effectiveness. The results show that how much improvement our proposed word-level N-gram gain over the previous character N-gram.

RQ2) *Under what conditions are the optimum experimental results achieved?* In the experiment, many parameters will affect the quality of the modeling and the experiments results, like the number of topics K , the N-gram parameter N and the thresholds. We want to explore the effect of varying the number of topics, N and thresholds to the performance of LNG.

RQ3) *Whether the combination of LDA and N-gram model results in better or poorer performance?* In this research question, we want to answer whether the LNG model is superior to the previous LDA and N-gram model. We compare the results of LDA, N-gram and LNG to demonstrate whether the combination of LDA and N-gram model performs better.

RQ4) *How much improvement could our LNG model gain over Wang XY's results, REP, BM25F and DBTM?* As to how well does our LNG model work compare with other methods, we compare our LNG model with Wang XY's results, REP, BM25F and DBTM. This question could be used to confirm how much improvement our LNG model could gain over the state-of-art approaches.

4.4 Results and Discussion

4.4.1 How Effective Is Our Proposed Word-Level N-Gram?

Figure 3 shows the recall rate of our proposed word-level N-gram model in comparison with Sureka's character N-gram on the Eclipse Data Set, where the x-axis indicates different values of k for Top- k results (top k bug reports descending ordered by similarity), and the y-axis indicates the corresponding recall rate. The parameter N for the N-gram in this experiment is set to 3. In the comparison experiment, we implement Sureka's N-gram [13] on our dataset with the same parameter setting of our approach.

From Fig. 3, we can see our proposed word-level N-gram achieves higher recall rate in comparison with

Fig. 4 The influence of the number of topics on the experiment results.

Sureka's character-level N-gram. We improved the result by approximately 3.4% on average.

4.4.2 Under What Conditions Are the Optimum Experimental Results Achieved?

The parameter selection will directly affect the quality of the modeling and the experiments results when building models for specific sample spaces. We ran LNG on Eclipse data set as topics number T was varied from 5 to 100 with the step of 5. Figure 4 shows that the influence of the number of topics on the experiment results with the parameter $N = 3$, threshold = 0.5. The precision rate increases with the number of topics at first, and then reaches a steady state later, while the recall rate overall is on a slow but steady downward curve, but the recall rate is always at a high level. Especially when the number of topics is 45, both the recall and precision rates reached 95.8%. Furthermore, the EA rate increases at first, reaches a peak at 45 topics and then declines.

From Fig. 4, we see that when there are too few topics (topics number $T < 30$), the precision rate is low. This is reasonable because the discrimination between topics is not obvious enough and the particle size distribution is insufficient, the number of features of bug reports is too small to distinguish their technical functions, thus, there are many documents classified into the same topic group even though they contain other technical topics. When the number of topics increases to a large size ($T > 60$), the recall rate becomes low, the reason might be because too many topics make it dispersive. It causes a document to have many topics with similar proportions which may lead to unsatisfying results. This overfitting problem degrades the precision rate. Therefore, setting the number of topics to a reasonable number ($30 < T < 60$) gives good experimental results. The reason might be because the number of topics in these ranges reflects well the numbers of technical issues in those bug reports.

Similarly, as shown in Fig. 5 which indicates the influence of the N value on the experiment results with the parameter topics number $T = 45$, threshold = 0.5, the precision rate increases along with the N value, while the recall rate declines. The EA rate also increases at first and then declines. When N is small ($N < 3$), the precision rate is low. This is reasonable because the N-gram items are too short

Fig. 5 The influence of the N value on the experiment results.

to distinguish their technical functions, which leads to much unsatisfying results. When N is 3 or 4, the recall rate and precision rate achieve reasonable values, especially when N is equal to 3, the experimental results achieve the highest correct results. When N is greater than 4, the precision rate stabilizes at 1, but the recall rate drops down sharply. The reason might be because N-gram items are too long to match, which leads to missing the correct results. This suggests that the parameter N should be set to a reasonable value ($N = 3$, or 4). This is also consistent with the typical values in most cases [47], [48]. Duplicate bug reports detection should first consider the recall rate, and then the precision rate, because the recall rate reflects the correct rate of the duplicate bug reports, which tells us whether this method is feasible, while the precision rate refers to the manual cost reduction rate in duplicates detection. This is just like cost-sensitive theory [49], which means the cost of erroneously diagnosing duplicate bug reports as non-duplicates may be much higher than that of diagnosing non-duplicate bug reports as duplicates. Diagnosing duplicate bug reports as non-duplicates signifies a reduction in the recall rate while diagnosing non-duplicate bug reports as duplicates signifies a reduction in the precision rate. The priority of the recall rate is higher than the precision rate. After overall consideration, we chose $N = 3$ as the optimal value, the recall rate, precision rate are at 95.8%, the EA rate also reaches the highest rate.

From Fig. 4 and Fig. 5, we can see that the EA rate curve is related with the recall rate and the precision rate curve. The first half of the EA rate curve is close to the precision rate curve; the second half is close to the recall rate curve. It expresses the shortcoming of recall rate and precision rate, which represent the weakness of a certain model to some extent. When the EA rate is high, means both the recall rate and precision rate are high, the model can be considered as an excellent model.

The setting of weight and threshold will also affect the experiment results. Since the weights of the LDA model and N-gram model are automatically updated to the optimum by machine learning, we just considered the threshold. Figure 6 shows the change of the recall rate, precision rate and EA rate using different thresholds with the parameter $N = 3$, topics number $T = 45$.

As shown in Fig. 6, when the weight is updated to the

Fig. 6 The influence of threshold on the experiment results.

Fig. 7 The influence of the weight of categorical information on the experiment results.

optimum by the adaptive adjustment algorithm, the recall rate as a whole decreases with the threshold. Thus, when the threshold is high (threshold > 0.6), the obtained results have a low recall rate, which is from the less experimental results. Both the precision rate and EA rate increase at first and then decline, which represents the threshold should be set to an appropriate value ($0.2 < \text{threshold} < 0.6$). In the experiments, the best threshold value is 0.5, which the recall rate and precision rate all reach 95.8%, the accuracy reaches 90.9%.

To explore the influence of the weight of categorical information on the experiment results, Fig. 7 shows the results with the parameter $N = 3$, topics number $T = 45$, threshold $= 0.5$. The weight of the categorical information is varied from 0 to 1 with the step of 0.1. As shown in Fig. 7, the recall rate as a whole increases with the weight of categorical information while the precision rate decreases. When the weight is 0, the categorical information is not used, and we obtain a low recall rate. When the weight of categorical information is 0.1, both the recall rate and precision rate are very high, reaching 95.8%. This suggests that the categorical information can help improve the performance of duplicate detection. When the weight is greater than 0.1, the recall rate is relatively high, but the precision rate and EA rate drop sharply. The reason might be because there are many recommended results which lead to more incorrect results. Only when the weight of categorical information is set to a reasonable value (0.1), both the recall rate and precision rate achieve the best results. Similar to Fig. 5, we choose 0.1 as the optimal weight of categorical information.

Fig. 8 The influence of the training data on the experiment results.

Fig. 9 Five number summary of words for bug reports in training data.

To explore the relation between the performance of LNG and the amount of bug reports, we first select 1538 bug reports from 2006.12.16 to 2007.12.15 as the test sample according to Sect. 4.1, and then divided the remaining datasets into 6 non-overlapping frames (or windows) by the creation time. Each frame contains 12 months bug reports, and the number of bug reports from frame 0 to frame 5 is 43,693, 46,876, 40,374, 32,606, 20,513, and 22,374 respectively. For example, frame 0 contains the bug reports generated from 2006.12.16 to 2007.12.15; frame 1 contains the bug reports generated from 2005.12.16 to 2006.12.15 et al.. The validation process proceeds as follows: First, we train using bug reports in frame 0 and test the bug reports in the test sample. Then, we train using bug reports in frame 0 and frame 1 and use the similar way to test the bug reports in the test sample and so on. In the final fold, we train using bug reports in frame 0-5 and test the bug reports in test sample. Figure 8 shows the experimental results with the parameter $N = 3$, topics number $T = 45$, threshold $= 0.5$. As shown in Fig. 8, the precision rate and the EA rate grow slightly with the increase of the size of the training data, while the recall rate fluctuates around 0.95. When training using bug reports in frame 0-5, the results achieve the best results. The reason might be because the large amount of training data constructs the LDA model well, the topics reflect well the technical issues in those bug reports.

Similarly, to explore the relation between the performance of LNG and the number of words, we visualize the five number summary of words for bug reports as shown in Fig. 9, where the x-axis is the training data, and the y-axis indicates the number of words. Further-

Fig. 10 The influence of the number of words on the experiment results.

Fig. 11 The experimental results using different group of test samples.

more, we count the sum of the number of words from frame 0 to frame 5, is 16,735,957, 16,367,508, 13,976,052, 12,027,232, 7,395,102, and 10,279,416 respectively. And then, we compute the average number of words (i.e., document size) for each group of training data. In detail, we sort the 6 groups of training data in ascending order, namely from group 0 to group 5. The number of words from group 0 to group 5 is 359.54, 361.3, 361.4, 365.51, 371.94 and 383.04 respectively. Figure 10 shows the influence of the number of words on the experiment results with the parameter $N = 3$, topics number $T = 45$, threshold = 0.5, where the x-axis denotes the different groups ascending ordered by the average number of words. As shown in Fig. 10, the recall rate grows slightly with the increase of the number of words while the precision rate fluctuates around 0.95. The reason for the increasing recall rate might be because larger document size means larger training data, which constructs the LDA model better, and the topics reflect the technical issues well.

Also, we randomly sampled 10 groups of bug reports as test samples to explore the stability of the LNG model according to Sect. 4.1, which contain 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000 duplicate bug reports respectively. We named them from group 0 to group 9. Figure 11 shows the experimental results. As shown in Fig. 9, we can see that the recall rate fluctuates with the test sample, but is relatively stable at a high rate. The precision rate always keeps the high rate on different groups of test sample.

As mentioned above, the different values of parameters affect the detection results. There is no provably optimal choice for the parameters [50], [51]. The choice is a trade-

Fig. 12 The comparison of LNG with LDA and N-gram.

off between coarser parameters (smaller value) and finer-grained parameters (larger value). So a reasonable choice should be made which is mainly dependent on the experiments and experience of the experimenter.

4.4.3 Whether the Combination of LDA and N-Gram Model Results in Better or Poorer Performance?

Since there is a question as to whether the LNG model is superior to the previous LDA and N-gram model, Fig. 12 shows the recall rate, precision rate, and EA rate of the LNG model, LDA model, and N-gram model with the parameter $N = 3$, topics number $T = 45$, the threshold = 0.5. From Fig. 12, we can see the LNG model can improve the performance than the related LDA or N-gram model, since its recall rate, precision rate and EA rate are all greater than the LDA or N-gram model. In addition, the results also show that the N-gram model performs better than the LDA model. Considering that the topic model LDA is suitable for large text, but bug reports are not always large, this may explain why the word-based model N-gram performs better than the topic model LDA. In the bug report detection field, word-based models are superior to topic models.

4.4.4 How Much Improvement Could Our LNG Model Gain over Wang XY's Results, REP, BM25F and DBTM?

Since the highest recall rate so far is Wang XY's results [4], which considered the execution information. The comparison of LNG with Wang X Y's result is shown in Fig. 13.

As shown in Fig. 13, the recall rate parallels that of Wang XY [4], but the precision is improved except the number of topics is 10. Compared with the traditional method which adopted to detect duplicate bug report with executive information, the recall rates are stable when using different number of topics, roughly the same at about 95%, but most of the precision rates are greater than Wang XY's results. As for this experiment, when 50 topics were selected, the pre-

Fig. 13 The comparison of LNG with Wang X Y.

Fig. 14 Recall result of LNG in comparison with other models.

recision rate was up to 95.8%, which was an obvious increase compared with 67% by the traditional method.

Since most of papers used recall rate to evaluate the experimental results, Fig. 14 displays the recall result of LNG in comparison with the state-of-art models for the Eclipse dataset, which are all the latest methods for detecting duplicate bug reports. To make a comparison to them, we used the same datasets with [7], [14] which contains 45234 bug reports from 2008.1.1 to 2008.12.30, so we used REP's result from [14] and BM25F, DBTM's results from [7] in Fig. 14. Since most of them used the top k list, we also used the top k list to replace the threshold in order to make a comparison to them.

As shown in Fig. 14, LNG achieves a very high recall rate, it is higher than other models [7], [14]. Our results show that there are near 6% higher than DBTM within a list of top-1 result, 4.5% higher than DBTM within a list of top-5 results and 3.5% higher than DBTM within a list of top-10 results. In comparison to this case, LNG achieves higher recall rate from 2.5%-6% with DBTM, 11%-18% with REP, and 21%-24% with BM25F. LNG can relatively improve DBTM by 2.96%-10.53%. REP and BM25F cannot detect the duplicate bug reports which are not textual similar. In our LNG model, the topic similarity from the

topic model LDA can help further improve the performance of the N-gram than those not textually similar fields. In addition, our results are also better than DBTM, because our LNG model considered the word order, contextual and categorical information which cannot be ignored for duplication detection.

5. Validity Threats

In our study, we assume that the information provided by the bug reporter is correct, including the textual, categorical, and duplication information. If a bug report information is not enough or misleading, the performance of LNG model is adversely affected.

The next validity threat is limited by the sole use of the Eclipse open source bug repository, but this is very large and popular with tens of thousands of developers worldwide, it contains more than 213,000 bug reports, which is one of the maximum quantity of bug reports processed for duplicates detection in one project in my own limited experience, and since Eclipse uses Bugzilla as its bug tracking system, which is the most widely used bug tracking system, the breadth of the Eclipse sub-projects provides some form of generality. In the future, we would consider more software systems, especially on commercial projects.

Another validity threat is the experimental parameters configuration. There is no general solution for choosing the optimal parameters, such as the number of topics K , N-gram parameter N , and the threshold. Even though we studied on the effects of different parameter settings and the models can yield acceptable results, we cannot ensure that our setting is optimal for all experimental datasets.

6. Conclusion

In this paper, we improved the N-gram similarity algorithm and proposed a method named LNG to detect duplicate bug reports in Bugzilla. We consider five factors including: textual, semantic, word order, contextual connection and categorical information. These features are fed to LNG built by combining the topic model, LDA, and the word-based model, N-gram. The LNG model performed well on the duplicate bug reports written with the same textual tokens as well as written with semantically similar ones. It is complementary to each other and also improves the robustness of duplicate bug reports detection. LNG models bug reports as textual documents, this method is not limited to detecting duplicate bug reports, it can also be applied to duplicate texts constructed by natural language in other fields.

We have also compared LNG with several state-of-art methods. The results on more than 213,000 Eclipse bug reports show LNG outperforms the previous LDA and N-gram model in terms of the recall rate, the precision rate and the proposed EA rate. Moreover, the recall rate has also improved by up to 10.52% compared to the DBTM approach.

Acknowledgments

The work described in this paper was partially supported by the National Natural Science Foundation of China (Grant no. 91118005, 61173131), Chongqing Graduate Student Research Innovation Project (Grant No. CYS15022), the China Postdoctoral Science Foundation under Grant 2014M560704, the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry and the Fundamental Research Funds for the Central Universities Project No. 2015CDJXY.

References

- [1] J. Anvik, L. Hiew, and G.C. Murphy, "Coping with an open bug repository," *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange - eclipse '05*, New York, New York, USA, pp.35–39, ACM Press, 2005.
- [2] L. Hiew, *Assisted detection of duplicate bug reports*, Ph.D. thesis, The University Of British Columbia, 2006.
- [3] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," *Software Engineering*, 2007. ICSE 2007. 29th International Conference on, pp.499–510, IEEE, 2007.
- [4] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," *Proceedings of the 30th international conference on Software engineering*, pp.461–470, ACM, 2008.
- [5] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," *Dependable Systems and Networks With FTCS and DCC*, 2008. DSN 2008. IEEE International Conference on, pp.52–61, IEEE, 2008.
- [6] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pp.45–54, ACM, 2010.
- [7] A.T. Nguyen, T.T. Nguyen, T.N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," *Automated Software Engineering (ASE)*, 2012. *Proceedings of the 27th IEEE/ACM International Conference on*, pp.70–79, IEEE, 2012.
- [8] G. Salton, *The SMART retrieval system-experiments in automatic document processing*, Prentice-Hall, 1971.
- [9] A. Alipour, A. Hindle, and E. Stroulia, "A Contextual Approach Towards More Accurate Duplicate Bug Report Detection," *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13*, Piscataway, NJ, USA, pp.183–192, IEEE Press, 2013.
- [10] Y. Tian, C. Sun, and D. Lo, "Improved Duplicate Bug Report Identification," 2012 16th European Conference on Software Maintenance and Reengineering, pp.385–390, IEEE, March 2012.
- [11] D.M. Blei, A.Y. Ng, and M.I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol.3, pp.993–1022, 2003.
- [12] G. Salton, A. Wong, and C.S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol.18, no.11, pp.613–620, Nov. 1975.
- [13] A. Sureka and P. Jalote, "Detecting Duplicate Bug Report Using Character N-Gram-Based Features," 2010 Asia Pacific Software Engineering Conference, pp.366–374, IEEE, Nov. 2010.
- [14] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), pp.253–262, IEEE, Nov. 2011.
- [15] C.-Z. Yang, H.-H. Du, S.-S. Wu, and X. Chen, "Duplication Detection for Software Bug Reports Based on BM25 Term Weighting," *Technologies and Applications of Artificial Intelligence (TAAI)*, 2012 Conference on, pp.33–38, IEEE, 2012.
- [16] N. Kaushik and L. Tahvildari, "A comparative study of the performance of ir models on duplicate bug detection," *Software Maintenance and Reengineering (CSMR)*, 2012 16th European Conference on, pp.159–168, IEEE, 2012.
- [17] J. Zhou and H. Zhang, "Learning to rank duplicate bug reports," *Proceedings of the 21st ACM international conference on Information and knowledge management*, pp.852–861, ACM, 2012.
- [18] J. Zhang, X. Wang, D. Hao, B. Xie, L. Zhang, and H. Mei, "A survey on bug-report analysis," *Science China Information Sciences*, vol.58, no.2, pp.1–24, 2015.
- [19] K. Aggarwal, T. Rutgers, F. Timbers, A. Hindle, R. Greiner, and E. Stroulia, "Detecting duplicate bug reports with software engineering domain knowledge," *Software Analysis, Evolution and Reengineering (SANER)*, 2015 IEEE 22nd International Conference on, pp.211–220, IEEE, 2015.
- [20] F. Thung, P.S. Kochhar, and D. Lo, "DupFinder: integrated tool support for duplicate bug report detection," *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pp.871–874, ACM, 2014.
- [21] A. Lazar, S. Ritchey, and B. Sharif, "Improving the accuracy of duplicate bug report detection using textual similarity measures," *Proceedings of the 11th Working Conference on Mining Software Repositories*, pp.308–311, ACM, 2014.
- [22] S. Rastkar, G. Murphy, and G. Murray, "Automatic Summarization of Bug Reports," *IEEE Trans. Softw. Eng.*, vol.40, no.4, pp.366–380, 2014.
- [23] S. Banerjee, B. Cukic, and D. Adjeroh, "Automated Duplicate Bug Report Classification Using Subsequence Matching," 2012 IEEE 14th International Symposium on High-Assurance Systems Engineering, pp.74–81, IEEE, Oct. 2012.
- [24] M. Amoui, N. Kaushik, A. Al-Dabbagh, L. Tahvildari, S. Li, and W. Liu, "Search-based duplicate defect detection: an industrial experience," *Mining Software Repositories (MSR)*, 2013 10th IEEE Working Conference on, pp.173–182, IEEE, 2013.
- [25] J. Lerch and M. Mezini, "Finding Duplicates of Your Yet Unwritten Bug Report," 2013 17th European Conference on Software Maintenance and Reengineering, pp.69–78, IEEE, March 2013.
- [26] R.P. Gopalan and A. Krishna, "Duplicate Bug Report Detection Using Clustering," *Software Engineering Conference (ASWEC)*, 2014 23rd Australian, pp.104–109, IEEE, 2014.
- [27] T. Zhang and B. Lee, "A Novel Technique for Duplicate Detection and Classification of Bug Reports," *IEICE Transactions on Information and Systems*, vol.E97.D, no.7, pp.1756–1768, 2014.
- [28] P. Hooimeijer and W. Weimer, "Modeling bug report quality," *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pp.34–43, ACM, 2007.
- [29] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?," *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pp.308–318, ACM, 2008.
- [30] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," *Software Maintenance*, 2008. ICSM 2008. IEEE International Conference on, pp.346–355, IEEE, 2008.
- [31] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan, "GATE: an architecture for development of robust HLT applications," *Proceedings of the 40th annual meeting on association for computational linguistics*, pp.168–175, Association for Computational Linguistics, 2002.
- [32] O. Gospodnetic and E. Hatcher, *Lucene*, Manning, 2005.
- [33] A.K. McCallum, "Mallet: A machine learning for language toolkit," University of Massachusetts, 2002.
- [34] D. Blei and J. Lafferty, "Text mining: Theory and applications, chapter topic models," 2009.

- [35] J.D. Mcauliffe and D.M. Blei, "Supervised topic models," *Advances in Neural Information Processing Systems*, pp.121–128, 2008.
- [36] S. Grant, J.R. Cordy, and D.B. Skillicorn, "Using topic models to support software maintenance," *Software Maintenance and Reengineering (CSMR)*, 2012 16th European Conference on, pp.403–408, IEEE, 2012.
- [37] S.W. Thomas, B. Adams, A.E. Hassan, and D. Blostein, "Validating the use of topic models for software evolution," *Source Code Analysis and Manipulation (SCAM)*, 2010 10th IEEE Working Conference on, pp.55–64, IEEE, 2010.
- [38] T.T. Nguyen, T.N. Nguyen, and T.M. Phuong, "Topic-based defect prediction (nier track)," *Proceedings of the 33rd international conference on software engineering*, pp.932–935, ACM, 2011.
- [39] Y. Fu, M. Yan, X. Zhang, L. Xu, D. Yang, and J.D. Kymer, "Automated Classification of Software Change Messages by Semi-supervised Latent Dirichlet Allocation," *Information and Software Technology*, vol.57, pp.369–377, 2014.
- [40] T. Hofmann, "Probabilistic latent semantic indexing," *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pp.50–57, ACM, 1999.
- [41] T.K. Moon, "The expectation-maximization algorithm," *Signal processing magazine, IEEE*, vol.13, no.6, pp.47–60, 1996.
- [42] G. Casella and E.I. George, "Explaining the Gibbs sampler," *The American Statistician*, vol.46, no.3, pp.167–174, 1992.
- [43] G. Heinrich, "Parameter estimation for text analysis," Technical report, Univ. Leipzig, 2008.
- [44] W.B. Cavnar, J.M. Trenkle, and Others, "N-gram-based text categorization," *Ann Arbor MI*, vol.48113, no.2, pp.161–175, 1994.
- [45] V. Keşelj, F. Peng, N. Cercone, and C. Thomas, "N-gram-based author profiles for authorship attribution," *Proceedings of the Conference Pacific Association for Computational Linguistics (PACLING)*, pp.255–264, 2003.
- [46] G. Kondrak, "N-gram similarity and distance," *String processing and information retrieval*, vol.3772, pp.115–126, Springer, 2005.
- [47] Y. Miao, V. Keselj, and E.E. Milios, *Comparing document clustering using n-grams, terms and words*, Master's thesis, Dalhousie University, 2004.
- [48] M. Shafiei, S. Wang, R. Zhang, E. Milios, B. Tang, J. Tougas, and R. Spiteri, "Document representation and dimension reduction for text clustering," *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pp.770–779, IEEE, 2007.
- [49] Y.F. Li, J.T. Kwok, and Z.H. Zhou, "Cost-Sensitive Semi-Supervised Support Vector Machine," *AAAI*, pp.500–505, 2010.
- [50] H.M. Wallach, I. Murray, R. Salakhutdinov, and D. Mimno, "Evaluation methods for topic models," *Proceedings of the 26th Annual International Conference on Machine Learning*, pp.1105–1112, ACM, 2009.
- [51] S.W. Thomas, B. Adams, A.E. Hassan, and D. Blostein, "Studying software evolution using topic models," *Science of Computer Programming*, vol.80, pp.457–479, 2014.

Jie Zou was born in 1992 and received his B.S. in Chongqing University in 2014. He is currently a M.S. candidate of the School of Software Engineering, Chongqing University. His research interests include mining software repositories, empirical study and topic modeling.

Ling Xu was born in 1975 and received her B.S. in Hefei University of Technology in 1998, and her M.S. degree in software engineering in 2004. She received her Ph.D. degree in computer science technology from Chongqing University, PR China in 2009. Her research interests include data mining of software engineering, topic modeling, and image processing.

Mengning Yang received the M.S. degree in Mathematics from Chongqing University, PR China in 2002, where he also received the B.S. degree in 2005. He is an Associate Professor in School of Software Engineering at Chongqing University. His current research interests data mining of software engineering and topic modeling.

Xiaohong Zhang received the Ph.D. degree in computer software and theory from Chongqing University, PR China in 2006, where he also received the M.S. degree in applied mathematics. He is a Professor in School of Software Engineering at Chongqing University. His current research interests include data mining of software engineering, topic modeling, image semantic analysis and video analysis.

Jun Zeng received the BS and MS degree in software engineering from Chongqing University in 2007 and 2010. He received Ph.D. degree in advanced information technology of Kyushu University in 2013. Since 2013, he has been a lecturer in the school of software engineering of Chongqing University. His research interest includes Web data extraction, data mining, and search engine.

Sachio Hirokawa received the BS and MS degree in Mathematics and PhD degree in Interdisciplinary Graduate School of Engineering Sciences from Kyushu University in 1977, 1979, and 1992. Since 1997, he has been a professor in research institute for information technology of Kyushu University. His research interest includes search engine, text mining, and computational logic.