Automatic Stop Word Generation for Mining Software Artifact Using Topic Model with Pointwise Mutual Information*

Jung-Been LEE (a), Student Member, Taek LEE (b), and Hoh Peter IN (c), Nonmembers

SUMMARY  Mining software artifacts is a useful way to understand the source code of software projects. Topic modeling in particular has been widely used to discover meaningful information from software artifacts. However, software artifacts are unstructured and contain a mix of textual types within the natural text. These software artifact characteristics worsen the performance of topic modeling. Among several natural language preprocessing tasks, removing stop words to reduce meaningless and uninteresting terms is an efficient way to improve the quality of topic models. Although many approaches are used to generate effective stop words, the lists are outdated or too general to apply to mining software artifacts. In addition, the performance of the topic model is sensitive to the datasets used in the training for each approach. To resolve these problems, we propose an automatic stop word generation approach for topic models of software artifacts. By measuring topic coherence among words in the topic using Pointwise Mutual Information (PMI), we added words with a low PMI score to our stop words list for every topic modeling loop. Through our experiment, we proved that our stop words list results in a higher performance of the topic model than lists from other approaches.

key words: text mining, software artifact, stop words, topic modeling, Pointwise Mutual Information (PMI)

1. Introduction

In the field of software engineering, mining software artifacts such as source code changes, bug databases, and commit messages from repositories is known as a useful way to understand and maintain the source code of software projects [1], [2].

Discovering and retrieving meaningful information from artifacts in the repositories is a difficult task as the artifacts are unstructured textual data, which makes analysis difficult. Thus, in many software artifact mining tasks, topic models have been used extensively to overcome this problem and effectively discover a set of topics within the data for various purposes (e.g., feature location [3], bug localization [4], [5], similarity visualization [6], and understanding functional intent, and an overview of the software [7], [8]).

Topic models like latent Dirichlet allocation (LDA) [10] are statistical models that serve as a kind of information retrieval method for mining large volumes of unstructured textual data. When the data have a clear topic and are well written (e.g., large collections of abstracts from scientific literature) [33], the generated topic model is usually easily understood and fit for its purpose. However, software artifacts are varied in terms of language construction (e.g., programming language syntax and the identifier) and contain a mix of textual type (e.g., natural text, source code, and stack traces in bug reports or commit messages) in the natural text. Additionally, the data is sometimes sparse or noisy and can be domain-specific. These characteristics make it difficult for topic modeling to generate a high-performing model because the modeling methods do not consider the type of textual data. Therefore, we require preprocessing of the data, which is called natural language preprocessing [11].

Typically, preprocessing includes the following steps: 1) tokenizing, 2) removing stop words, 3) stemming and lemmatization, and 4) decapitalizing letters. In particular, the second step is an efficient way to improve the quality of a topic model because the step performs a key role in removing meaningless and uninteresting terms called stop words. Conventionally, stop words lists, extracted through many approaches such as RAKE [12], Poisson distribution [19], and Fox stoplist [27], have been widely used in text analysis. However, these lists are outdated and too general for standalone or off-the-shelf use with domain-specific documents [23] like software artifacts. For example, “apache” is not a stop word in classical lists, but the term would be a stop word in software using the Apache Commons library developed by the Apache Software Foundation because the keyword “apache” would exist in most of the source code. Alternatively, “problem” or “problems” can be stop words in a bug database from a software repository, but they are not included in classical lists. In the case of our software artifact textual dataset, nearly 97–99% of terms in the Fox stoplist [27] are not used as stop words. Through our experiment, we found that the performance of the approach (i.e., RAKE [12]) using stop words generated from pretrained textual dataset are sensitive to the kind of dataset. Therefore, the stop words list generated by these approaches such as RAKE, Poisson distribution, and Fox stoplist, does not perform well in other software artifacts that were not used in training. The results tell us that these stop words lists are inappropriate for use with software artifacts.

In this paper, we propose an automatic stop word generation approach (AutoGen) for topic models of software artifacts. To accomplish this, we use Pointwise Mutual Infor-
mation (PMI) to measure topic coherence among the most probable words in a topic. In our approach, commit messages from the past four years are used as external data reflecting current domain information. The word association score between all pairs of words in the top topic words is measured by PMI, and the final word with the lowest sum of scores is automatically added to our stop words list. Through comparative evaluation, the stop words list generated by our proposed approach improves the performance of the topic model from about 14 to 124% with only 106 and 75 words in comparison with other stop words lists based on the Fox stoplist, Poisson distribution and RAKE algorithm.

The remainder of this paper is organized as follows. Section 2 introduces related work and a context for our approach, such as topic modeling and PMI. Section 3 presents our proposed approach and Sect. 4 introduces the experimental setup and performance measurements. We evaluate the experimental results by performing a case study and discussions in Sect. 6. Section 7 concludes the paper and outlines our future research directions.

2. Related Work and Context

2.1 Stop Word Generation

In information retrieval using textual analysis, a stop words list is based on common function words and is manually written by text analysts for their application, domain or specific languages to improve textual analysis [11], [12].

In early studies of word frequencies in a textual collection, Luhn [17] found that most discriminant words appear in the middle of term frequency (TF). Salton [18] extended the foundation further using document frequency (DF), and he proposed a measure of the degree of significance required for a word to be considered a stop word. He used the product of the term frequency and the inverse document frequency ($tf \cdot idf$), which means that stop words have high document frequencies or low term frequencies. Based on this measurement, Fox [27] presented a stop words list extracted from the Brown Corpus of English literature, and the list has been widely used in textual analysis. However, as we mentioned before, the lists were outdated and did not reflect the latest terms used in current articles, news, blogs, and tweets. We used the Fox stoplist as a baseline for comparative evaluation.

Another recent study proposed an unsupervised method to make stop words list [20]. The method generates stop words list for Polish texts based on the idea that common stop words follow a Poisson distribution [19]. According to the study, the DF of words in a textual corpus can be estimated (DFE: document frequency estimation) from their TF. Therefore, randomly distributed stop words in a textual corpus is defined by the Poisson distribution (see Eq. (1)), where N is the total number of documents and $P(0, \mu)$ is the probability of the word occurring 0 times:

$$DFE = N\left(1 - P\left(0, \frac{TF}{N}\right)\right) \quad (1)$$

The definition can be simplified to Eq. (2), and we apply this equation in Sect. 4.6 as one of approaches to generate a stop words list:

$$DF = N\left(1 - \exp\left(-\frac{TF}{N}\right)\right) \quad (2)$$

If the DEF/DF of some words is close to 1, the words are randomly distributed terms, i.e. stop words, and highly cluttered terms can be keywords to be eliminated in the stop words list. However, this approach is too general to consider the textual corpus depending on the domain and language because it does not consider the semantic relationships between the textual corpus but is based on statistics using Poisson distribution.

Recently, a domain- and language-independent method was proposed called RAKE (Rapid Automatic Keyword Extraction) [12]. In the first stage of this approach, keywords from individual documents are extracted through an unsupervised method. In the next stage, a supervised method generates a stop words list from a textual corpus, based on accumulating adjacency frequency (AF) and keyword frequency (KF) with TF and DF. Under the assumption that words adjacent to keywords tend to be stop words, words with $AF > KF$ are selected as stop words. We also generate stop words list using the approach in Sect. 4.6 as a representative approach with similar benefits to our approach.

2.2 Topic Modeling

Topic models, such as Latent Semantic Indexing (LSI) [9] and latent Dirichlet allocation (LDA) [10], [25], are statistical models that are types of information retrieval methods for mining large volumes of unstructured textual data (i.e., natural language from a variety of types of text, such as news articles, blogs, twitter, etc.). LDA is a probabilistic statistical model for collections of textual documents, represented by the bag-of-words model. It estimates the distributions of latent topics in textual data.

In a topic model, each document in document collection D is modeled as a multinomial distribution over K topics. Additionally, each topic is a multinomial distribution over W words. A “topic” is composed of a group of words that frequently occur together. Topic models can link words with similar meanings and distinguish between the uses of words with multiple meanings [21].

In many software engineering tasks, including mining software repositories, topic models have been widely used to discover a set of topics within the data for various purposes (e.g., feature location [3], bug localization [4], [5], similarity visualization [6], and understanding functional intent and an overview of the software [7], [8]). Because topic modeling is an unsupervised method, it makes the tasks easy to perform in practical settings. In addition, topic models do not require expensive data collection and preparation costs because they use unstructured text.

In our study, the words observed from the top-ranked topics in the topic model are used to generate a stop words
list because those words are considered more representative than the other words for the given topic. The words are ranked according to the probabilities obtained from the training phase, and the highly-ranked words are more likely to appear in the topic than the other words. Most software engineering researchers and analysts examine only these words to identify the latent topic from the results of their topic models. We call these words “associated words” in our study. An associated words list exists for every topic in a topic model. If a topic model has five topics, the number of associated words lists are also five. In Sect. 3.5, we express the associated words lists for each topic as associated words\#1, associated words\#2, …, associated words\#k. However, it is not easy to interpret and understand latent topics by simply exploring the given associated words because they usually look noisy. Therefore, to make it easier to adopt the associated words, we measure the coherence level[14], [33] between the words in each topic and then chose a stop word candidate for the given topic. These tasks are repeated for all topics, and a final stop word is chosen at each stage of the automatic generation loop.

2.3 Pointwise Mutual Information (PMI)

PMI has been studied in the context of co-occurrence of a word pair [26]. Generally, PMI measures statistical independence by observing two words in external textual data to evaluate the topic model. However, in our study, we measure the PMI score of each word pair in the topic model from the collection of documents itself because the purpose of our study is to generate stop words in the topic model.

Similar to the approach for topic model evaluation [13], we count co-occurrences with each other of associated words in the list for each topic and to compute the PMI score of a given word pair \((w_i, w_j)\). The PMI score is defined as follows (see Eq. (3)):

\[
\text{PMI}(w_i, w_j) = \log \frac{P(W_i, W_j)}{P(w_i)P(w_j)}
\]  

(3)

where \(p(w_i)\) and \(p(w_j)\) are the probabilities of occurrence of the words in the collection of documents. The probabilities can be represented as follows (see Eq. (4)):

\[
p(w) = \frac{TF(w)}{N}
\]

(4)

where \(TF(w)\) is the term frequency of word\(w\) in the collection of documents and \(N\) is the total number of documents. Additionally, the \(p(w_i, w_j)\) can be represented as follows (see Eq. (5)):

\[
p(w_i, w_j) = \frac{CO(w_i, w_j)}{N}
\]

(5)

where \(CO(w_i, w_j)\) is the number of co-occurrences of \(w_i\) and \(w_j\) in the collection of documents.

3. Automatic Stop Word Generation

This section presents an overview of our proposed automatic stop word generation process, which is as follows:

1) Textual data collection, 2) Counting term frequencies, 3) Topic modeling of the textual data and generating associated words list, 4) Measuring quality of topic model, 5) Measuring Word PMI (stop word candidates), 6) Measuring Topic PMI, and 7) Adding the final stop word to the list. Figure 1 provides an overview of our approach.

The conditional loop from Steps 3 to 7 (dotted grey box) is continually repeated until the stop words list is optimized. In other words, the loop is ended when several loop conditions are satisfied, as defined in Step 3. The number of stop words depends on the type of textual data and software project domain.

3.1 Step 1: Textual Data Collection

To mine software artifacts in the field of software engineering, the related textual data are collected by the software repository. As mentioned in Sect. 1, the data can be source code and its changes, bug databases, email archives, execution logs, commit messages, and so forth [2]. The collected data plays a role in our approach as an external dataset that reflects domain knowledge and the characteristics of the software artifacts, as mentioned above in Sect. 1.

According to the type of the data, proper preprocessing should be considered. For example, for source code, some tasks to extract identifier names, comments and string literals should be performed. In addition, for bug reports, a task should be performed to separate natural text, source code and stack trace. The preprocessed data are merged into a text corpus for Steps 2 and 3.

3.2 Step 2: Counting Term Frequency

PMI requires two variables, such as the term frequency (TF) number and co-occurrence of word pairs (CO) in a text corpus [13], [14]. In this step, we count these variables in a text corpus collected from Step 1.

TF is counted by the occurrence of a certain word\(w\) in the text corpus. If the word “apache” occurred three times in the text corpus, the word is marked “3”. CO is the accumulated frequency of the co-occurrence of word \(i(w_i)\) and word
Table 1  Term and co-occurrence frequency

<table>
<thead>
<tr>
<th>word j (w_j)</th>
<th>TF(w_j)</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>apache</td>
<td>16,026</td>
<td>15,289</td>
</tr>
<tr>
<td>edef</td>
<td>15,363</td>
<td></td>
</tr>
<tr>
<td>contribute</td>
<td>16,026</td>
<td>9,374</td>
</tr>
<tr>
<td>the</td>
<td>18,640</td>
<td>6,991</td>
</tr>
<tr>
<td>block</td>
<td>2,190</td>
<td>7,868</td>
</tr>
<tr>
<td>hdfs</td>
<td>1,582</td>
<td>7,868</td>
</tr>
<tr>
<td>https</td>
<td>15,706</td>
<td>6,454</td>
</tr>
<tr>
<td>when</td>
<td>15,706</td>
<td>1,814</td>
</tr>
<tr>
<td>https</td>
<td>15,706</td>
<td>1,001</td>
</tr>
<tr>
<td>changes</td>
<td>15,706</td>
<td>1,814</td>
</tr>
<tr>
<td>hdfs</td>
<td>15,706</td>
<td>1,001</td>
</tr>
<tr>
<td>fix</td>
<td>23,050</td>
<td>1,376</td>
</tr>
<tr>
<td>window</td>
<td>23,050</td>
<td>500</td>
</tr>
<tr>
<td>hadoop</td>
<td>21,045</td>
<td>1,379</td>
</tr>
<tr>
<td>report</td>
<td>21,045</td>
<td>200</td>
</tr>
<tr>
<td>create</td>
<td>2,435</td>
<td>3,599</td>
</tr>
<tr>
<td>move</td>
<td>1,411</td>
<td>111</td>
</tr>
</tbody>
</table>

j(w_j) in each document. If the word “apache” and “hadoop” occurred concurrently in five documents, the pair of words is marked “5”. Table 1 shows an example of the TF and CO results from our textual corpus, sorted by CO value.

As shown in Table 1, “apache” occurred 16,026 times and “edef” occurred 15,363 times in the text corpus, respectively. Additionally, these two words appear together in 15,289 documents.

For comparative evaluation, in Sect. 5.2, variables related to other studies such as classical lists (i.e., Fox stolplist), Poisson-based stoplists, and RAKE stoplists are also calculated.

3.3 Step 3: Topic Modeling

In this step, topic modeling is performed with the text corpus. In the first stage of the automatic stop word generation loop, topic modeling is performed using an empty stop words list. The empty stop words list is filled with final stop words, one at a time, in Steps 3.3 to 3.7. The output of this stage is associated words for topics without any stop words.

The following words are a sample output of topic modeling in this step.

- Topic1: branch, hadoop, common, merge, org, ffa, ...
- Topic2: fix, add, message, update, error, server, ...

Topic modeling algorithms such as LDA order associated words by their weight, which refers to the usage frequency of the corresponding associated word for each topic. In the sample previously given, the words “branch” and “fix” occur more frequently than do other words in Topics 1 and 2, respectively. To mine software artifacts, a text analyst identifies latent topics from the aforementioned topic groups and finds additional informative characteristics for his or her purpose in the software artifacts involved in the topic group.

3.4 Step 4: Measuring Quality of the Topic Model

After topic modeling, we measure the perplexity of the topic model to find the optimal number of stop words. Perplexity is a widely used evaluation metric to evaluate the quality of the topic model, where a lower perplexity score indicates that the model is a better-quality model.

The measure is the perplexity of held-out documents, which is a decreasing function of log-likelihood (\(= L(w)\)) of the \(w_j\), as expressed in the following equation; we used a script for calculating perplexity score in the Git repository:

\[
\text{perplexity (test dataset } w \text{)} = \exp \left\{ - \frac{L(w)}{\text{count of tokens}} \right\}
\]

The measure is the perplexity of held-out documents, which is a decreasing function of log-likelihood (\(= L(w)\)) of the \(w_j\), as expressed in the following equation; we used a script for calculating perplexity score in the Git repository:

\[
\text{perplexity (test dataset } w \text{)} = \exp \left\{ - \frac{L(w)}{\text{count of tokens}} \right\}
\]

Figure 2 shows an example case of perplexity scores calculated for incremental stop words up to 421 ones; that is, it is the result of our loop process repeated from Steps 3 to 7. The horizontal axis represents the number of stop words generated by our proposed process, and the vertical axis represents the perplexity score.

An accurate number of stop words improves the topic model; furthermore, stop words can degrade the model quality. Therefore, we need a heuristic to set the loop boundary where we can find the optimal perplexity score. In our approach, linear regression analysis is used to predict the negative and positive slope trends of perplexity scores. The perplexity scores repeatedly calculated in each loop are used as the independent variable in regression analysis and the slope of the coefficient is then obtained to presume the trend of perplexity scores. The moment when the slope turns over from negative (−) to positive (+) is considered as the loop end point (the loop escaping condition) and we then search for the minimum perplexity score over the negative slope area. In Fig. 2, the collection area collects the least number of samples and boots up the regression analysis. The other negative and positive slope areas need to be determined to obtain the minimum perplexity score and loop end point. The two parts of the negative and positive slope areas, shown in Fig. 2, are not determined by a regression analysis, but by a simple snapshot of local data over the horizontal axis after the collection area. Rather, the signs of the negative or positive slopes are determined by a regression analysis with the overall data accumulated from the beginning to the end of the decision-making point (e.g., the negative slope area of Fig. 2 was determined by a regression analysis with data collected from 1 to 297 on the horizontal axis).

†https://github.com/Mondego/LDA-studies/blob/master/Mallet/scripts/calculate-perplexity.sh
Therefore, the loop escapes when the number of stop words reaches 297 and the optimal number of the stop words observed is 90. The reason why we do not take the positive slope area into account is that the perplexity scores tend to show an increasing trend over the area. In Eq. (6), the numerator \( L(w) \) is converged to a certain minus value (log of probability) [10], [41] and the dominator “count of tokens” simultaneously decreases consistently as the number of stop words increases.

### 3.5 Step 5: Measuring Word PMI Score

This step aims at finding the word list of a stop word candidate by measuring the PMI score of all pairs of the associated words. As the source variables of TF and CO of the associated words were already counted in Step 2, we generate the PMI matrix for each topic \( k \) to find a stop word candidate in each topic \( k \) as shown in Table 2, where \( n \) is the number of associated words. Although in practice, the number \( n \) can be chosen based on the size of the vocabulary, many studies [5], [8], [10], [13]–[15], [31]–[33] have typically analyzed the top 5-to-20 associated words and considered them to be sufficient to capture some of the underlying topics in the corpus. Determining the number \( n \) might entail a cost-benefit analysis between the costs and benefits. An illustrated cost-benefit analysis is presented in Sect. 4.4.

As shown in Table 2, the PMI score can be presented as \( N \times N \) symmetric matrix because PMI(\( w_i, w_j \)) and PMI(\( w_j, w_i \)) have the same score, where \( i \) and \( j \) represent the row and column of the table, respectively. If the number of topic \( K \) is 10, 10 symmetric matrices are also generated.

<table>
<thead>
<tr>
<th>associated word#1</th>
<th>associated word#2</th>
<th>associated word#3</th>
<th>\ldots</th>
<th>associated word#n</th>
</tr>
</thead>
<tbody>
<tr>
<td>( PMI(w_{i1}, w_{j1}) )</td>
<td>( PMI(w_{i1}, w_{j2}) )</td>
<td>( PMI(w_{i1}, w_{j3}) )</td>
<td>\ldots</td>
<td>( PMI(w_{i1}, w_{jn}) )</td>
</tr>
<tr>
<td>( PMI(w_{i2}, w_{j1}) )</td>
<td>( PMI(w_{i2}, w_{j2}) )</td>
<td>( PMI(w_{i2}, w_{j3}) )</td>
<td>\ldots</td>
<td>( PMI(w_{i2}, w_{jn}) )</td>
</tr>
<tr>
<td>( PMI(w_{i3}, w_{j1}) )</td>
<td>( PMI(w_{i3}, w_{j2}) )</td>
<td>( PMI(w_{i3}, w_{j3}) )</td>
<td>\ldots</td>
<td>( PMI(w_{i3}, w_{jn}) )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>\ldots</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( PMI(w_{in}, w_{j1}) )</td>
<td>( PMI(w_{in}, w_{j2}) )</td>
<td>( PMI(w_{in}, w_{j3}) )</td>
<td>\ldots</td>
<td>( PMI(w_{in}, w_{jn}) )</td>
</tr>
</tbody>
</table>

In this step, a \( N \times m \) matrix for topic \( k \) is created to measure the PMI score between the candidate words selected from Step 5 and the associated words of each topic, for \( K \) topics. Table 3 represents the matrix for topic \( k \).

<table>
<thead>
<tr>
<th>candidate word#1</th>
<th>associated word#1</th>
<th>associated word#2</th>
<th>\ldots</th>
<th>associated word#n</th>
</tr>
</thead>
<tbody>
<tr>
<td>( PMI(c_{i1}, w_{j1}) )</td>
<td>( PMI(c_{i1}, w_{j2}) )</td>
<td>( PMI(c_{i1}, w_{j3}) )</td>
<td>\ldots</td>
<td>( PMI(c_{i1}, w_{jn}) )</td>
</tr>
<tr>
<td>( PMI(c_{i2}, w_{j1}) )</td>
<td>( PMI(c_{i2}, w_{j2}) )</td>
<td>( PMI(c_{i2}, w_{j3}) )</td>
<td>\ldots</td>
<td>( PMI(c_{i2}, w_{jn}) )</td>
</tr>
<tr>
<td>( PMI(c_{i3}, w_{j1}) )</td>
<td>( PMI(c_{i3}, w_{j2}) )</td>
<td>( PMI(c_{i3}, w_{j3}) )</td>
<td>\ldots</td>
<td>( PMI(c_{i3}, w_{jn}) )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>\ldots</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( PMI(c_{in}, w_{j1}) )</td>
<td>( PMI(c_{in}, w_{j2}) )</td>
<td>( PMI(c_{in}, w_{j3}) )</td>
<td>\ldots</td>
<td>( PMI(c_{in}, w_{jn}) )</td>
</tr>
</tbody>
</table>

Finally, a word with minimum SPMI among \( n \) number of associated words in each topic should be selected as a stop word candidate. If \( K \) is 10, \( 10 \) words are selected in total for all topics. Equation (8) is defined to select stop word candidates for each topic number \( k \) in the topic models. At the end of this step, \( k \) number of words are selected as our stop word candidates.

\[
\text{Candidate}(k) = \min_{i=1,\ldots,n} \{\text{SPMI}_k(w_i)\}, \quad (8)
\]

If the two words (\( w_i \) and \( w_j \)) never co-occur (i.e., \( \text{CO}(w_i, w_j) = 0 \)) in the PMI calculation, the PMI score is minus infinity according to Eq. (3). To avoid the exception, we set the PMI score to zero in our study. If all PMI scores are zero in the calculation of SPMI(\( w_i \)), the SPMI(\( w_i \)) score is also zero. If there are more than two such cases that have an SPMI score of zero in a single topic, we choose the word having the larger denominator (\( = P(w_i) \times P(w_j) \)) in Eq. (3).

### 3.6 Step 6: Measuring Topic PMI Score

In this step, we select a single word as the final stop word to be added to our stop words list. Although the candidates selected in Step 5 have the minimum PMI score in their own topic, they may have associations with words in other topics. Thus, in this step, we again measure all pairs of candidates extracted during the process described in Sect. 3.5 with the top associated words chosen for each topic from the process described in Sect. 3.3.

In this step, a \( n \times m \) matrix for topic \( k \) is created to measure the PMI score between the candidate words selected from Step 5 and the associated words of each topic, for \( K \) topics. Table 3 represents the matrix for topic \( k \).

As for the word PMI, if the number of topics, \( K \), is 10, 10 matrices for each topic are generated. Then, the SPMI(\( c_{i} \)) of the candidate word \( n \) is calculated by summing all columns consisting of the given candidate word \( n \) in each matrix for topic \( k \). To decide on the final stop word, the SPMI(\( c_{i} \)) of each topic are accumulated as a SPMI(\( c_{i} \)) and a final stop word is decided by selecting the candidate word with the smallest value among the accumulated SPMI(\( c_{i} \)) scores. Namely, if the number of topics is 10, 10 accumulated SPMI(\( c_{i} \)) scores are calculated and only one candidate word is selected among them as a final stop word.

### 3.7 Step 7: Adding Final Stop Word to List

The final stop word chosen in Step 6 is added to our stop words list, and the list is used for topic modeling in Step 3 again. As a result, the number of stop words depends on the number of automatic stop word generation loops (Step 3~Step 7). Because the generated stop words do not appear again in the topic model during the next loop, new
words are consistently added to this stop words list.

4. Experimental Setup

This section presents our experimental setup and measurements to compare the performance of the ascertained topic model between our approach and other methods.

4.1 Collected Dataset

We collected textual datasets from software artifacts of open-source projects on a Git repository. We chose the actively developed or developing projects in popularity order (called Stargazer in GitHub) with the “Java” languages option. In particular, we collected commit messages related to bugs from among the software artifacts by searching using keywords such as “bug” and “fix” [24]. The reason we collected the software artifacts is that many studies [1], [2], [38]–[40] in the field of software engineering analyze commit messages for bug localization, bug triage, and prioritization.

To acquire enough data for topic modeling, we collected the bug-related commit messages for the four years between 2012 to 2015. Table 4 presents the projects selected for our case study and their data properties. As shown in the table, the number of commit messages is approximately over 5,000, and the textual size of commit messages is over 250 kb. Because the Actor project started October 12, 2014, the collection period is only about 1 year. However, the number and size of commit messages are large enough for our experiment.

In our approach, the commit messages of all of the projects were used as a source of the training dataset during training of the topic model (Step 3). Some of the training datasets were used to evaluate the quality of the topic model in computing the perplexity and generating the stop words list that were necessary prior to the training of the topic model. For this purpose, we randomly chose 30 different samples from the training dataset (Step 4). We called the dataset the ‘Internal’ dataset. Putting aside the training dataset, the other test dataset, the ‘External’ dataset, was necessary to validate the effectiveness of our stop words list. This dataset had to be unseen to the already trained model. Therefore, the future data of the commit messages from all of the projects were collected from 2016 for test purposes. The number of commit messages collected from 2016 was 44,778, which is approximately 30% of the total number for the training datasets. We utilized 30 samples from the External dataset and measured their marginal probabilities for the performance evaluation in the case study of Sect. 5.2.

4.2 Preprocessing for Textual Corpus

In order to count term frequency (TF) and co-occurrence (CO), we performed preprocessing on the textual corpus collected in Sect. 4.1. First, all the commit messages collected from the 12 projects were combined into a single textual corpus. Second, some NLP (Natural Language Processing) was performed, such as removing punctuation, composing words under a two-letter alphabet and decapitalization, excluding removing stop words.

4.3 Optimal Number of Topics Selection

We ran topic modeling using Mallet tool†† as a form of LDA implementation. The tool is a Java-based machine learning for language toolkit, and the tool provided a Java API to import data, train topic models, and infer topics for new documents. We did not control the alpha and beta hyper parameters of topic models but used the default options provided by the tool.

There are some methods [25], [28]–[30] that are widely used in information retrieval (IR) for selecting the number of topics with the best performance. In our study, we used the minimization approach [29] (i.e., CaoJuan2009) to select the optimal number of topics before generating the optimal stop words list. The method provides a metric to adaptively find the optimal number of topics based on topic density. According to the result of the method, the lowest value means that the LDA model has the best performance for the number of topics. Figure 3 shows the calculated values according to the number of topics from our training datasets.

From the results, we can conclude that optimal number

---

Table 4 Projects for case study and its commit information

<table>
<thead>
<tr>
<th>Project</th>
<th># of Commit</th>
<th>Commit size</th>
<th>Collection period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>4,897</td>
<td>255 KB</td>
<td>10/16/2014–12/31/2015</td>
</tr>
<tr>
<td>Alltoio</td>
<td>8,297</td>
<td>560 KB</td>
<td>12/22/2012–12/31/2015</td>
</tr>
<tr>
<td>Cassandra</td>
<td>7,149</td>
<td>897 KB</td>
<td>12/15/2011–12/30/2015</td>
</tr>
<tr>
<td>CoreNLP</td>
<td>12,617</td>
<td>641 KB</td>
<td>06/27/2013–12/30/2015</td>
</tr>
<tr>
<td>Druid</td>
<td>4,947</td>
<td>402 KB</td>
<td>10/24/2012–12/30/2015</td>
</tr>
<tr>
<td>Graylog</td>
<td>8,676</td>
<td>688 KB</td>
<td>01/02/2012–12/23/2015</td>
</tr>
<tr>
<td>Hadoop</td>
<td>21,801</td>
<td>3.75 MB</td>
<td>01/01/2012–12/30/2015</td>
</tr>
<tr>
<td>Kotlin</td>
<td>29,334</td>
<td>2.01 MB</td>
<td>12/21/2011–12/30/2015</td>
</tr>
<tr>
<td>Netty</td>
<td>8,400</td>
<td>1.90 MB</td>
<td>01/07/2012–12/30/2015</td>
</tr>
<tr>
<td>OrientDB</td>
<td>9,073</td>
<td>550 KB</td>
<td>01/03/2012–12/30/2015</td>
</tr>
<tr>
<td>PDE</td>
<td>6,948</td>
<td>360 KB</td>
<td>01/05/2012–12/20/2015</td>
</tr>
</tbody>
</table>

††http://mallet.cs.umass.edu/
of topics is 50. Therefore, we set the optimal number of topics as 50.

However, the methods based on held-out likelihood or statistics of the whole corpus are generally intended to set a large number of topics because they do not consider the internal representation of the models and do not examine the structure of the topics themselves. The increasing number of topics worsens the interpretation of the topics and does not increase the agreement between human subjects and the model [31] accordingly. In software analysis, because approximately 10–25 topics yield valuable results [32], we also set the number of topics to 10 as the representative number of topics for our study.

4.4 Number of Associated Words Determination

Consistent with the existing studies mentioned in Sect. 3.3, we did not consider more than 20 associated words for calculating the PMI score in Step 5. Nevertheless, a procedure for deciding the best number of associated words would be useful. A small number will generate a poor quality of a topic model and a large number will require long computing times for the PMI scores. Thus, an analysis should be performed of the trade-off between the benefits and costs.

Figure 4 presents the results of a cost-benefit analysis between the quality of the topic model and time complexity of calculating the PMI score in Step 5. Nevertheless, a procedure for deciding the best number of associated words would be useful. A small number will generate a poor quality of a topic model and a large number will require long computing times for the PMI scores. Thus, an analysis should be performed of the trade-off between the benefits and costs.

Figure 4 presents the results of a cost-benefit analysis between the quality of the topic model (= Benefit), the calculation time required to measure the PMI score (= Cost), and the Trade-off point based on the Trade-off curve. The Benefit is the value of the accumulation probability of the associated words in topics of the topic model, which is an output result of the Mallet tool when using the “--topic-word-weights-file” option. The Cost is the time complexity of measuring the PMI score, i.e. the time required to calculate an n×n matrix for each topic, as presented in Table 2, where n is the number of associated words. For convenience in the calculation of Trade-off values, we normalized the Cost and Benefit to [0, 1]. The Trade-off value was calculated using following equation:

\[
\text{Trade-off value} = \text{Benefit} \times (1 - \text{Cost})
\]  

(9)

As shown in the figure, the Benefit curve is rising and the Cost curve continuously decreases as the number of associated words reaches 20. Although the greater the number of associated words, the more beneficial they are for interpreting the topic, the cost to calculate the PMI score increases simultaneously (on the right-hand side of the vertical axis the cost increases). In the meantime, the Trade-off point suggests the best number of associated words, considering both the ease of topic interpretation and time cost. In our experimental setting, we assumed this number was seven as shown in Fig. 4. However, the decision of the associated words number is not only limited to the method suggested in this section, but rather it rests with the analysts. If more time is allowed to prolong the number searching process, all of the 20 associated words and perhaps even more can be considered.

4.5 Evaluation Metric

The estimated probability of held-out (i.e. test) data is a clear and interpretable metric for evaluating the quality of topic models relative to other topic models. Among the metrics for estimating the probability, we employed Marginal probability (Wallach’s “left-to-right (L2R)” algorithm [16], [34]), which is widely used in the evaluation of topic models [31], [35]–[37] because it provides a clear methodology for accurately assessing and selecting topic models, can be generalized easily, and allows for likelihood-based comparisons of different models or the selection of model parameters.

The algorithm is implemented in MarginalProbEstimator class of MALLET tools and the class is able to run using the MALLET command option “evaluate-topics”. We measure the marginal probability of topic models learned from the training dataset using each stop word list for the test dataset. The pseudo code for evaluating the stop words list is described in Fig. 5.

In the code, we create 30 samples of test datasets to measure the marginal probability (See Sect. 4.1). In the loop for calculating the marginal probability, each stop word list (= STOPWORDLISTS[j]) is used to perform topic modeling from training datasets. After training the topic model, the marginal probability (= probability) for each test dataset (= testSets[j]) is calculated using MALLET’s ‘evaluate-topics’ option in the second loop. Finally, the total sum of the probabilities (= totalSum) for test datasets are averaged by the total number of test datasets using the averageProbability variable and compared among the stop words lists.

4.6 Extracting Stop Words of Other Approaches

The Fox stoplist serves as a baseline for our study and includes 425 stop words. The list does not require extra tasks to extract words.

To extract stop words from the Poisson approach, we set the threshold for the related DFE/DF. Following the approach introduced in [11], we used the Fox stoplist that appears in the training dataset to compute the mean and standard deviation of the DFE/DF. Thus, we calculated the mean...
Fig. 6 Perplexity of the number of stop words in topic model with 10 topics

Fig. 7 Perplexity of the number of stop words in topic model with 50 topics

Fig. 5 Pseudo code used in our study to calculate the marginal probability of DFE/DF + standard deviation of DFE/DF (= 1.12) and obtained 5,544 stop words that appear in at least 10 documents (DF > 10) in our training dataset. The number of words was about 10% (5,544/55,549) of all words in the textual corpus.

We also extracted stop words using the RAKE algorithm. To extract keywords, we used the implementation of RAKE in the Git repository (https://github.com/aneesha/RAKE). From these keywords, we count keyword frequencies (KF) and adjacent frequencies (AF), and we obtain 400 stop word candidates appearing in at least 10 documents as in the Poisson approach.

5. Case Study

To demonstrate the effectiveness of our approach for improving the quality of the topic model in software analysis, this section presents our evaluation results for measuring the performance of a topic model generated by each stop words list.

5.1 Selection of Optimal Stop Words

Our stop words list was adaptively generated depending on the domain, volume, and type of document (e.g. software artifacts), unlike constant stop words lists such as the Fox stoplist. Specifically, the list generated in this case study is not the absolute result for all software artifacts and even other documents similar to our dataset. Therefore, this step for selecting the optimal number of stop words is a prerequisite and should be performed in advance to apply our approach in practice. To perform the process, a single machine Intel® CPU i7-7700K CPU (4 cores) overclocked at 4.6 GHz with 16 GB memory was used. The operative system running in the performing machine was Windows 10 Pro. Python version was 2.7.13; and Mallet version was 2.0. We performed topic modeling by Mallet with a default parameter, except for the set the topic numbers.

Figures 6 and 7 show the perplexity curve per the number of our generated stop words when we generated the topic model that set the number of topics to 10 and 50 topics from the Internal test dataset. In each list, stop words were generated when the loop end condition was met, as we explained in Sect.3.4. As a result, our process generated 265 stop words in each set of 10 and 50 topics. The total elapsed time to generate 265 stop words was 8.6 hr (= 512 min). Namely, the time of one cycle to generate a stop word for a loop from steps 3 to 7 in our process took approximately 116 s on average. Among the steps, the performing topic modeling was the most time consuming. The elapsed time can be affected by many factors, such as the type of software artifacts (e.g., commit messages, source code, and stack traces...
in bug reports), quality of the topic model in step 4, size of datasets, hardware resources, and option settings of the Mallet tool. For example, reducing a value of the “--num-iterations” command option of the Mallet tool can save the time taken to complete sampling. Once a stop words list is generated, the process does not need be repeated, as for other approaches.

The solid line represents the average perplexity from 30 samples of each stop word in the optimal stop words list through our approach. The dotted line represents the perplexity measured from random stop words lists at each stop word. The random stop words lists were produced by randomly shuffling words from all stop words in the optimal stop words list generated through our approach (e.g., 265 stop words in the topic model with 10 and 50 topics). Then, we measured the perplexity from 0 (e.g., empty stop words) to the total number of stop words. We measured the perplexity 30 times and averaged the values to obtain the common result. The result of the random stop words list show that our optimal stop words list was not obtained by chance.

As shown in Fig. 6, the trend of the perplexity score dropped until the number of stop words reached 106, and the score is the minimum score for through the whole graph at the number of stop words. After the drop, the perplexity score of the Optimal stop words list gradually increased such that it approached the last stop word. Therefore, for the topic model with 10 topics, we chose these 106 stop words as the Optimal list for comparing performance with other approaches.

In the result of perplexity as shown in Fig. 7, the score by Optimal stop words list also dropped until the number of stop words reached 75. Thus, we chose 75 stop words with the lowest score in the graph as the Optimal list.

However, the perplexity scores of the random stop words list with both 10 and 50 topics were steadily higher than the Optimal stop words list through most of the graph. Although the score of the random stop words list was lower than that of our optimal stop words list in certain areas, the score was not lower than the minimum score of our optimal stop words list. Figure 8 shows the differences in perplexity that was selected as the minimum score of the Optimal and Random stop words lists from the topic model for 10 and 50 topics.

In the Topic 10 group, the perplexity score of the Optimal list is significantly lower (approximately 9.52%) than the score of the Random list as a result of t-test (p-value = 0.0001 ≤ 0.05), where the mean scores of the Optimal and Random lists are approximately 15,823 and 17,487, respectively. In Topic 50, the score of the Optimal list is significantly lower (approximately 8.15%) than that of the Random list (p-value = 0.0018 ≤ 0.05), where the mean scores are approximately 14,665 and 15,966 in the Optimal and Random list, respectively.

5.2 Evaluation for Topic Model Performance

This section compares the performance of a topic model using each approach. As mentioned in Sect. 4.6, other stop words lists are already generated using the Poisson approach and the RAKE algorithm. Fox stoplists are also used, as in the study [13].

Figure 9 shows the comparison results of the normalized marginal probability estimation values for each approach with the External dataset. The dataset was not used to generate stop words lists for these approaches, as mentioned in Sect. 4.1. As shown in Fig. 9, the list of our approach outperformed the other approaches for the External dataset in both of two topic groups.

The average probability estimation values in Topic 10 for AutoGen, Fox stoplist, Poisson, and RAKE were approximately 0.65, 0.46, 0.29, and 0.45, respectively. In Topic 10, our stop words list improved the performance of the topic model by approximately 41%, 124%, and 44% compared with other approaches, and a t-test indicated that the result was statistically significant as all the p-values were under 0.05. In Topic 50, the average scores of AutoGen, Fox stoplist, Poisson, and RAKE were approximately 0.49, 0.42, 0.32, and 0.43, respectively. Also, AutoGen improved the performance of the topic model by approximately 17%, 53%, and 14%. In addition, the p-values of other approaches were less than 0.05 according to t-tests.

Although the Fox stoplist has general words, without domain-specific words, in contrast with other approaches,
it is similar to RAKE, excluding ours. This means that the general English stop words are effective even in the topic model trained using commit messages. One the other hand, Poisson has a lower performance than other approaches for both topics 10 and 50. This is because it does not consider the semantic relationships between the textual corpus unlike our approach, but depends on probability distributions, as mentioned in Sect. 2.1.

5.3 Analysis of Stop Words

Our stoplists for the topic model with Topic 10 and Topic 50 include only 106 and 75 words, respectively, while the Fox stoplist and RAKE have over 400 words and Poisson has 5,544 words. Figure 10 illustrates the total ratio of the number of stop words provided by each approach.

As shown Fig. 10, the Fox stoplist and RAKE include less than 1% of the stop words in the textual corpus. However, our stop words list generates a more effective model with even fewer words. In the Poisson approach, the stop words are nearly 10% of the total number of words in the textual corpus because there is no pre-processing and there is a small threshold to filter the words. Also, the ratio of our stop word numbers overlapped by 3, 2, and 5% with the stop words lists of the Fox stoplist, Poisson, and RAKE approaches, respectively. Namely, a tiny proportion of our stop words appear in other stop words lists.

For further detail, Table 5 shows some the overlapped words and checks whether the word occurs in the list for the External dataset.

In the case of the Fox stoplist, our stop words include the following overlapping words in the list: use, this, now, with, for, next, from, group, only, was and, which. For our textual corpus, only these words are meaningful as stop words. The other words are not used (e.g., outdated) or have no relationship with keywords generated by topic modeling. RAKE stop words list, based on keywords as in our approach, found the following words written in commit messages that are domain related but not included in keywords: reviewed, git, mapreduce, beta, apache, master, web, fixes and so forth. Although the performance is higher than all other approaches except our approach, this approach focuses on adjacency to a keyword but does not consider the relationship among all words in a document.

Poisson distribution based stop words include the highest number of words when compared to other approaches. However, the approach relies solely on probability distribution without considering the domain of the document and the relationships between words.

6. Discussion

Although our approach improves the performance of topic models by measuring marginal probability, the interpretation of the topics may not necessarily be correlated with the measurement [21]. Therefore, in this section, we discuss this issue and look at the topic model from the perspective of human interpretation.

Table 6 lists the top ten associated words with an Empty stop words list for External dataset. Because the stop words list did not contain any words, many noisy words appeared in every topic. For example, the, that, all, some, only, and, but, when, which, can, are, was, have, should, will, not, more, about, into, with, than, before, and after as classic stop words appeared in all topics. In particular, Topic 8 was mainly composed of classic stop

---

**Table 5** A part of overlapped words in each list and the occurrence

<table>
<thead>
<tr>
<th>Word</th>
<th>Fox stoplist</th>
<th>Poisson</th>
<th>RAKE</th>
</tr>
</thead>
<tbody>
<tr>
<td>reviewed</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>use</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>fixes</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>modification</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>for</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>git</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>apache</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>with</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>now</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>contribute</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>mapreduce</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>lucu</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>https</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>master</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>motivation</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>result</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>conflicts</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
</tbody>
</table>

**Table 6** Associated words in each topic with empty stop words list for External dataset

<table>
<thead>
<tr>
<th>Topic#</th>
<th>Associated words</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>the and use when can, which using not used more</td>
</tr>
<tr>
<td>2</td>
<td>with type methods name property object instead compiler parameters move</td>
</tr>
<tr>
<td>3</td>
<td>ifa edef fixes fails change should block page cluster after</td>
</tr>
<tr>
<td>4</td>
<td>and the with http add support client when not data</td>
</tr>
<tr>
<td>5</td>
<td>build api version files configuration dependency about module cache notes</td>
</tr>
<tr>
<td>6</td>
<td>java into commit fix merge master changes request xml com</td>
</tr>
<tr>
<td>7</td>
<td>and code method default add remove instead value null removed</td>
</tr>
<tr>
<td>8</td>
<td>the that are not all but have only than path</td>
</tr>
<tr>
<td>9</td>
<td>the when not was will after before time and which</td>
</tr>
<tr>
<td>10</td>
<td>test tests some with issue and task more case output</td>
</tr>
</tbody>
</table>
words. In addition, several overlapped words were present in the topics. Therefore, we could not classify the topics and identify a latent topic in the topic model. Table 7 shows the associated words with an Optimal stop words list for the External dataset.

Although most of the classic stop words in Table 6 still existed, including the and with, some topic interpretations became only slightly clearer than the results given in Table 6. For example, words in Topic 10 of Table 6 and Topic 1 of Table 7 seemed to indicate a correlation with the “test” topic. In Table 6, some unrelated words such as issue, and many classic stop words such as and, some, with, and more interrupted the interpretation of the topic, whereas all of classical words were removed excluding for; and related words such as build, configuration, dependency, and module were newly added to the top 10 list in Table 7. Because, overall, the words in the topic were condensed toward the “test” topic, the topic interpretation with our Optimal stop words list became clearer. Furthermore, we were able to find another latent topic, namely, “bug,” in Topic 6 from the newly added words such as fixes, issue, and bug.

7. Conclusion
Removing the stop words task in natural language preprocessing can help to improve the quality of textual analysis methods such as topic modeling. However, in the field of software repository mining, textual data such as software artifacts have different language construction and a mix of textual types in the natural text. The data may even depend on specific domain. The characteristics of software artifacts decrease the quality of a topic model.

Many studies for generating stop words have been applied to topic modeling to improve the quality of the models. However, the stop words lists contain words that are outdated when used with current types of documents (e.g., news, articles, twitter, and blogs). We proved this through our experiment, which found that nearly 97 to 99% of words in the classical lists were useless as stop words. Additionally, stop words lists generated from general literature do not reflect enough domain specific words. Therefore, the list is not suitable for our textual data, such as software artifacts.

In this paper, we proposed an automatic stop word generation approach for improving the quality of a topic model. The approach generates stop words by measuring topic coherence using a PMI score between all pairs of words in the topic model. We collected software artifacts such as commit messages from software repositories (i.e., Git repository) and used them for training the topic model clearly than. We applied marginal probability for held-out data as a widely used performance measurement in topic modeling and compared our stop words list with existing stop words lists extracted from the Fox stoplist, Poisson approach, and RAKE algorithm.

As the comparative evaluation shows, our stop words list improves the performance of the topic model from about 14 to 124% for the External datasets, relying on only 1% of the stop words in the textual corpus used for the training model. As a result, our approach helps generate the topic model to predict test data with high performance. However, the best performance with marginal probability does not always guarantee the best interpretation of the associated words for the relevant latent topics. Nevertheless, we were able to identify some latent topics more clearly than empty stop words lists.

For future studies, we plan to extend our approach to different types of software artifacts, such as code changes, bug databases, and code frequencies. In addition, we will add a process to prioritize our stop words to identify their optimal order. Through this process, our objective is to remove useless words and provide compact stop words list.

References


Jung-Been Lee is a Ph.D Candidate in the Department of Computer Science and Engineering at Korea University in Seoul, Korea. His major areas of study are software architecture evaluation and mining software artifacts & analysis. He received the M.S. degrees in Computer Science and Engineering from Korea University in 2011.

Taek Lee is currently an assistant professor in Department of Convergence Security Engineering at Sungshin University in Seoul, Korea. He received his Ph.D. in Computer Science and Engineering at Korea University in 2016. His research interests include software analytics, software defect prediction, mining software repositories, healthcare-ICT convergence, and information security.

Hoh Peter In received his Ph.D. degree in Computer Science from the University of Southern California (USC). He was an Assistant Professor at Texas A&M University. At present, he is a professor in Department of Computer Science and Engineering at Korea University in Seoul, Korea. He is an editor of the EMSE and TIES journals. His primary research interests are software engineering, social media platforms, and software security management. He earned the most influential paper award for 10 years in ICRE 2006. He has published over 100 research papers.