Abstract—Bug reports can be lengthy due to long descriptions and long conversation threads. Automatic summarization of the text in a bug report can reduce the time spent by software project members on understanding the content of a bug report. Our work further examines Rastkar et al.‘s use of a logistic regression model to determine which sentences from the text of a bug report should be extracted for creating a summary. Using their publicly available bug report corpus, which contains manually annotated bug reports, we examined two aspects regarding the features used by the model. First, we examined how much of a reduction occurs in the precision and recall if some of the more complex features are not used. Second, we examined how the use of different feature combinations affects the precision and recall of the models. We found that the absence of some of the complex features resulted in a modest decrease in precision and recall, and confirmed that some features, such as sentence length, were the most significant features for bug report summarization.

Index Terms—Bug reports, text summarization, software engineering, natural language processing

I. INTRODUCTION

Bug reports are useful software project artifacts that contain information about problems occurring in the past, discussion of possible or implemented solutions, and who contributed to these solutions. Bug reports contain a variety of textual information including a title, description, and comments [1]. The number of sentences in a bug report can vary widely, with some bug reports containing relatively few sentences (around 20) and some having over 50 sentences.

Bug report triage is an important software maintenance task where a project member examines each bug report and makes decisions about how the report will be handled [1]. Examples of bug report triage decisions include determining the quality, priority or severity of the report, assignment to a developer, validation of the report, and identification of duplicate reports. As software projects receive many bug reports each day [2], to effectively triage a new bug report, the person performing triage, called a triager, needs to understand the content of the new bug report, as well as previous bug reports. If a bug report is long (i.e. a long description and/or many comments), the triager needs to spend a significant amount of time reading and understanding the content. Presenting a summarized version of a bug report has been proposed to reduce the time taken by triagers in examining new or existing bug reports and has been shown to reduce the time for understanding the content without necessarily reducing the meaning of the original bug report [3], [4]. By reading the summarized version, a triager could more quickly triage a bug report, such as determining whether it is a duplicate of an existing bug report or whether it needs to be fixed. Bug report summarization can also benefit developers, as they could quickly get an overview of how a particular or related problem was handled in the past.

As the content of a bug report is primarily free-form text, natural language processing techniques, specifically those for text summarization, are used [3], [5]. There are two types of text summarization techniques: abstractive and extractive. Abstractive summarization requires understanding the content of the document and paraphrasing the content in such a way that the meaning is preserved. Extractive summarization selects sentences from the document that reflect the overall meaning of the document [6]. Text summarization uses both statistical and linguistic techniques to find the meaning and importance of a text.

Rastkar et al. [3] presented an extractive approach to bug report summarization that appears to work well. Their approach selects sentences to be extracted from the bug report based on a variety of features and a linear regression model. However, some of the features used in their approach are non-standard (e.g. clue word score), and extracting those features is complex. In this work, we investigate whether reasonable extractive summaries of bug reports can be created without the use of the complex features, and how the different features contribute to the creation of a reasonable extractive summary.

We begin by presenting an overview of previous bug report summarization work. We then present an overview of the data set and the extractive bug report summarization technique used in our investigation. Next, we describe how the summarizer was evaluated and the results of our investigation. We conclude the paper with a discussion of our findings.
II. RELATED WORK

We present in this section an overview of previous work related to bug report summarization.

Murray and Carenini [7] proposed an approach for extracting four types of features to summarize email and meeting conversation data. The extracted features were related to the length and the structure of the sentences, the lexical weights based on the conditional probabilities of each word’s appearance, and the participants of the conversation.

Rastkar et al. [3], [8] observed that comments in bug reports were similar to the conversation structure found in emails and meeting transcripts. Therefore they used the same feature extraction techniques as Murray to train a logistic regression model to classify sentences as being included in an extractive summary or not. They created an annotated bug report corpus for their investigation.

Lotufo, Malik, and Czarnecki [4] conducted a study of automatic, unsupervised bug report summarization that took a different approach than Rastkar et al.. Instead of using gold standard summaries, their study considered the similarity between a sentence and the bug report title, the similarity between two sentences, and the use of a heuristic to measure the agreement of two sentences. These characteristics were used to compare the quality of generated summaries. This approach led to a more generalized approach to evaluating bug report summaries.

Mani et al. [5] also proposed an unsupervised approach to text summarization. Instead of using the gold standard summaries, they used well known general purpose textual summarizers to create bug report summaries. However, they found that these summarizers only worked well when there was no noise present in the bug reports. To reduce the noise, they used heuristics to categorize text as being either a question, an investigative sentence or a code snippet.

III. BUG REPORT SUMMARIZATION

As previously stated, the goal of our study is to investigate the use of different features for extractive summarization of bug reports as presented by Rastkar et al. [3]. They categorized the 24 features into four sets: sentence length, lexical features, structural features and features related to the participants of the conversation in the bug report. As Rastkar et al. found that the F-score statistics of the structural and participant-related features have low variability, we focused our study on the length and lexical features. We also chose to remove the complex features of clue-word-score and those related to sentence entropy to investigate how extractive summaries selected without the use of these features compare to manually created summaries. In short, we investigated the use of two sentence length features, six conditional probability scores and four cosine similarity scores. The details for these features are explained in the following two sections.

A. Data Source

We used the bug report corpus created by Rastkar et al. [3] in our investigation. The bug report corpus consists of thirty-six (36) bug reports extracted from four open source software projects: Mozilla\(^3\), KDE\(^4\), Eclipse\(^5\) and Gnome\(^6\). Each bug report in the corpus has a title which indicates from which software project it comes. Each bug report’s comments are given in a format such as that of two or more people taking turns when having a conversation. Therefore, each comment is considered a turn and each turn has the participant’s name, the time when the conversation started, and the text of the individual sentences from the entire comment. Each bug report has two or more people participating in the conversation. Bug reports are stored in an XML format, each identified by a unique number.

The corpus also contains annotations made by three different annotators for each comment indicating whether the sentence should be included in an extractive summary. Following the procedure given by Rastkar et al. [3], we created gold standard summaries (GSS) for each bug report by including a sentence in the extractive summary if at least two annotators indicated it should be included in the bug report summary.

B. Extracted Features

1) Sentence Length: The two sentence length features extracted from each bug report comment are SLEN and SLEN2. SLEN is the length of a sentence normalized by the length of the longest sentence in all of the comments in the bug report. SLEN2 is the length of a sentence normalized by the length of the longest sentence in the specific bug report comment.

2) Lexical: In our investigation, ten lexical features were extracted from each sentence in a bug report comment. These lexical features are based on two different conditional probabilities known as $S_{prob}$ (for sentence probability) and $T_{prob}$ (for turn probability).

Equation 1 defines $S_{prob}$. Given a term $t$ by a person making a comment, $S_{prob}$ is calculated by finding the maximum probability of that term’s appearance in sentences from all of the comments $S$. For example, assume there are three commenters for a bug report: A, B and C. If A used the word $w_1$ seven times, B used $w_1$ twice and C used $w_1$ once, then the maximum probability of $w_1$ is 0.7, and all instances of $w_1$ in the bug report receive 0.7 as their $S_{prob}$ weight.

$$S_{prob}(t) = P(S|t)$$ (1)

Equation 2 defines $T_{prob}$. Given a term $t$ by a person making a comment, $T_{prob}$ is calculated by finding the maximum probability of that term’s appearance in a comment $T$. For example, if a bug report has five comments and the word $w_2$ appears eight times in one comment, twice in another comment, and in no other comments, then $T_{prob}(w_2)$ is 0.8.

---

\(^3\)bugzilla.mozilla.org, verified 09/12/2018
\(^4\)bugs.kde.org, verified 09/12/2018
\(^5\)bugs.eclipse.org/bugs, verified 09/12/2018
\(^6\)bugzilla.gnome.org, verified 09/12/2018
\[ T_{\text{prob}}(t) = P(T|t) \]  

(2)

Using each conditional probability weight we calculated three sentence level conditional probability features for the sum of weights, maximum weight and mean of weights. For \( S_{\text{prob}} \) the features calculated were named as SMS, MXS and MNS for sum, max and mean respectively. Similarly, for \( T_{\text{prob}} \) the names were SMT, MXT and MNT.

We calculated four cosine similarity scores for each sentence using \( S_{\text{prob}} \) and \( T_{\text{prob}} \) as the word encoding. COS1 and COS2 represent the sentence wise cosine similarity using \( S_{\text{prob}} \) and \( T_{\text{prob}} \) respectively. CENT1 and CENT2 represent the similarity of a sentence to the entire conversation using \( S_{\text{prob}} \) and \( T_{\text{prob}} \) respectively.

C. Bug Report Summary Creation

A logistic regression model was used to train a recommender which classifies each sentence from the bug report comments as appearing or not appearing in the extractive summary. The Python \texttt{sklearn} package was used to implement the sentence classifier. Various logistic regression models were trained to test the effect of the individual feature types and the combined effect of the length and lexical features regarding the performance of the classifier.

Following the procedure outlined by Rastkar et al., a bug report summary was created by selecting the sentences with the highest probability of being included in the extractive summary according to the classifier until the word count of the constructed summary reached 25% of the original bug report’s word count.

IV. Evaluation

To compare our model with Rastkar et al. [3] we trained a logistic regression model using our twelve selected features. We also investigated the effect of using only one of the two feature categories and different feature combinations. To investigate the effect of different feature combinations, we paired similar features into six groups (length, sum, max, mean, cos, cent) and created logistic regression models with combinations of the different groups. For example, choosing one group at a time creates six models (i.e. \( \binom{6}{1} \)) and choosing two groups at a time creates fifteen models (i.e. \( \binom{6}{2} \)). We created a total of sixty-two (62) models by choosing all combinations of the six groups (i.e. \( \binom{6}{1} \) to \( \binom{6}{5} \)).

We used the metrics of precision, recall and F-score to evaluate the different models.

Precision measures how many of sentences in the extractive summary were correct. It was calculated as the total number of sentences correctly classified as being in the extractive summary divided by the total number of sentences in the extractive summary.

\[ \text{Precision} = \frac{\text{# of sentences correctly selected}}{\text{total # sentences in the summary}} \]  

(3)

Recall measures how close the generated summary is to the GSS summary. It was calculated as the total number of sentences correctly classified as being in the extractive summary divided by the total number of sentences which appeared in the GSS summary.

\[ \text{Recall} = \frac{\# \text{ of sentences correctly selected}}{\text{total # sentences in GSS summary}} \]  

(4)

The F-score value is the harmonic mean of precision and recall.

\[ F - \text{score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \]  

(5)

As we used a leave-one-out cross-validation approach to evaluate each model, we received thirty-six (36) data points for the precision and recall of each model. We applied a post-hoc pairwise statistical test\(^7\) to determine if there was any statistically significant variation between the models.

V. RESULTS

In this section, we present a comparison of our extractive summarization approach with Rastkar et al.’s, the results of our investigation of using only sentence or lexical features, and the results of our investigation of combinations of our selected features.

A. Comparison to the Previous Approach

When we compare the results of our logistic regression model with a 25% threshold with that of Rastkar et al.’s, we found that our model has an average precision and recall that is less than theirs (Table I). However, this decrease is to be expected, given that fewer features are used in our model.

Our comparison with Rastkar et al.’s approach prompted us to further examine the contribution to extractive summarization of the individual features and their combinations.

| \text{Comparison of Our Results with Rastkar’s Results} |
|---|---|
| \text{Our model} | \text{Rastkar’s model} |
| \text{Precision} | 44% | 57% |
| \text{Recall} | 24% | 35% |
| \text{F-Score} | 29% | 40% |

B. Feature Categories

We examined the use of only sentence length features and only lexical features for creating an extractive bug report summarizer.

It was found that when using only the sentence length features (SLEN and SLEN2), the classifier had an average precision of 60% and the recall was found to be very low at 18%.

Using only the lexical features (SMS, MXS, MNS, SMT, MXT, MNT, COS1, COS2, CENT1, CENT2), we found that

\(^7\)Statistical analysis was conducted using R 3.5.1.
the recall improved to 22%, but the precision declined to 42%. The model which combined both the length and lexical features had a precision of 44% and a recall of 24%. As shown in Table II, the F-score was found to improve from 27% to 30% after adding the lexical features.

<table>
<thead>
<tr>
<th>Feature set</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>60%</td>
<td>18%</td>
<td>27%</td>
</tr>
<tr>
<td>Lexical</td>
<td>42%</td>
<td>22%</td>
<td>27%</td>
</tr>
<tr>
<td>All</td>
<td>44%</td>
<td>24%</td>
<td>29%</td>
</tr>
</tbody>
</table>

C. Feature Combinations

To understand the combined contribution of the features and to discover if there was a difference in the performance if we chose different combinations of features, we created sixty-two (62) different models by choosing different group combinations.

When testing for significance in the differences between combinations of features, only the sentence length features in (6) were found to be significant. The p-values for these comparisons are shown in Table III.

<table>
<thead>
<tr>
<th>Feature combinations</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLEN-SLEN2 vs. CENT1-CENT2</td>
<td>0.0062577</td>
</tr>
<tr>
<td>SLEN-SLEN2 vs. COS1-COS2</td>
<td>0.0004581</td>
</tr>
<tr>
<td>SLEN-SLEN2 vs. MNS-MNT</td>
<td>0.0013389</td>
</tr>
</tbody>
</table>

VI. DISCUSSION AND CONCLUSION

In our study, we focused on two types of features: sentence length and lexical features. According to Rastkar et al. [3] SLEN was one of the most important features when training an extractive bug report summarizer because of its high variability. Two lexical features SSM and TSM and the sentence length feature SLEN2 were also considered important. However, they also felt that all 24 features contributed to the overall performance of their model regardless of some of the features having less variability. Our results seem to corroborate this belief with our statistical analysis for different feature combinations, finding no statistically significant difference in performance when adding or removing features.

In training the logistic regression model with the sentence length and lexical features, we found that lengthy sentences often contained useful information needed to create a good summary. This is consistent with the literature which states that the sentence length is used to eliminate short sentences from the summary that do not contain useful information, such as author names or code extractions [9]–[11]. However, we also found that only using the length of the sentence was not enough to capture all of the useful sentences, as selecting longer sentences resulted in quickly reaching the word percentage threshold. Our results for the use of lexical features also show that the recall is higher when the length is not considered, as the classifier selects more of the shorter sentences thereby increasing the recall.

Finally, as we are taking a supervised learning approach to summary creation, the results are sensitive to the summaries found in the GSS. When comparing the word count of the summaries in the GSS for each bug report, we found that some summaries had a word count of more than 50% of the original bug report’s word count. As we used a word count threshold of 25% of the original bug report’s word count for the summaries, our model is unlikely to choose all the sentences found in the GSS summary for some of these summaries and this results in a low recall for the trained models.

We plan to further explore the use of the more complex features that have high variability and were not included in this study, namely clue-word score and those related to sentence entropy.

REFERENCES