Mining Intentions to Improve Bug Report Summarization

Beibei Huai,Wenbo Li,Qiansheng Wu,Meiling Wang
University of Chinese Academy of Sciences
Institute of Software, Chinese Academy of Sciences
National Engineering Research Center of Fundamental Software,Beijing, 100190, China
{beibei, wenbo, qiansheng, meiling}@nfs.iscas.ac.cn

***Abstract*—In recent years, various automatic summarization techniques have been proposed to extract important information from bug reports. However, existing techniques mainly focus on common text features and ignore human intentions implied in bug reports. In fact, each bug report generally contains multiple intentions which are distributed in different sentences. Bug report readers are usually more interested in content that contains intentions of certain categories (e.g. fix solution, bug description). Based on the above observation, we introduce an intention taxonomy and implement the intention classification algorithm in this paper. Furthermore, we propose a new Intention-based Bug Report Summarization approach, namely IBRS, which leverages intention taxonomy to enhance bug report summarization. We evaluate our approach on Intention-BRC corpus and the experimental result shows that IBRS outperforms the state-of-the-art approaches in terms of precision, recall, F-score, and pyramid precision.**

***Index Terms*—Bug Report, Text Summarization, Intention Taxonomy, Intention Mining**

I. Introduction

A software project's bug report repository provides a rich source of information for a software developer working on the project. A bug report is composed of a title, descriptions and comments from several developers. We analyse 40,000 bug reports from different open source projects and find that bug reports are usually very long (more than 10 comments in one bug report on average) and their content is mixed with code and debugging information. An example bug report is shown in Figure 1. It is part of the #434108 bug report of Eclipse which contains 14 comments. In such a long bug report, it is time-consuming for a reader to grasp the important information they need.

One of the most effective ways to save bug report readers' time is to provide them with summaries of bug reports. The main idea of the state-of-the-art bug report summarization techniques is to extract useful sentences in bug reports. Rastkar et al. [1] first proposed BRC model. They marked 36 bug reports (BRC corpus) and trained 3 classification models on BRC corpus, meeting corpus and email corpus to score each sentence in a bug report. Finally, they found that the result is sensitive to the type of corpus. It suggests that the text of bug reports has unique characteristics compared with other common texts. Mani et al. [2] used an unsupervised method.

10.18293/SEKE2018-096

Bug 434108- [Perspectives] Copy Worbench Layout option does not work in Eclipse 4.x

Status: ASSIGNED **Product:** Platform **Component:** UI (show other bugs)
Version: 4.4 **Hardware:** PC Windows 7 **Importance:** P3 normal (vote)
Assignee: Bartosz Popiela CLA Friend
Reported: 2014-05-05 08:31 EDT by Wojciech Sudol CLA Friend
Modified: 2017-05-26 09:23 EDT (History)

Description **Wojciech Sudol CLA Friend 2014-05-05 08:31:14 EDT**

Scenario:
1. Run Eclipse with a fresh workspace
2. Close Welcome page
3. Move and resize some views
4. From the main menu choose: File > Switch Workspace > Other...
5. In the "Workspace Launcher" dialog provide a path to a new workspace; select "Copy Settings" > "Workbench Layout" option and press OK.
6. After restart close Welcome page and compare new layout with the previous one.

Expected behaviour:
New workbench have the same layout as the old one.
Current behaviour:
New workbench have a new, default layout.
It was working in 3.x, does not work in 4.x.

Comment 1 **Eric Moffatt CLA Friend 2014-05-05 15:14:06 EDT**

Wojciech, this may not be perfect but it's at least working somewhat. I just opened a new workspace from my existing dev environment and it seems to have moved everything over (at least I started with both a Java and Debug perspective open and my EGit views where I expected them... This is in

Comment 2 **Eric Moffatt CLA Friend 2014-05-05 15:16:28 EDT**

I tested this on M7 and it's certainly copying most of the environment. I did the 'switch' from my regular dev workbench and the new workspace came up with both a Java and a Debug perspective and my EGit views where where I expected them.
What system are you on ?

Comment 3 **Wojciech Sudol CLA Friend 2014-05-05 15:38:37 EDT**

My OS is Windows 7 x64.
I was testing it again and I see that it works if you switch to already existing workspace, but not for new workspaces. In 3.x it works in both cases.

Fig. 1. A Bug Report Sample from Eclipse.

They classified the sentences roughly and dropped two kinds of the sentences (Question and Code). Finally they applied several unsupervised methods such as Grasshoppe, Diverse Rank to get summaries of bug reports. The sentences classification in their paper plays a role as a noise reduction mechanism. Moreover, the concept of intention has been applied to various software artifacts, e.g. app reviews [3], development emails [4]. We observe that sentences posted in bug reports also contain different purposes.

In this paper, we define 7 intention categories in bug reports and then explore the connection between summaries and sentence intentions in bug reports. Finally we improve the bug report summarization approach by taking sentences intentions into account. This paper mainly makes following 3 contributions:

1) We introduce an taxonomy of intentions in bug reports,

which classifies intentions into seven categories: *Bug Description*, *Fix Solution*, *Opinion Expressed*, *Information Seeking*, *Information Giving*, *Meta/Code* and *Emotion Expressed*.

2) We propose a mixed model of linguistic patterns matching and machine learning to classify the sentences of a bug report by their intentions.
3) We propose an intention-based bug report summarization approach(IBRS), which takes advantage of the informations that intentions implies.

The rest of this paper is organized as follows. Section II describes the related work. Section III gives an overview of IBRS approach. In Section IV, we describe the experiments we conducted and the evaluation results. We present threats to validity and future work in Section V. And we concludes our work in Section VI.

II. RELATED WORK

Summarization techniques are mainly classified into two categories: extractive [5] approach and abstractive approach [6], [7]. In recent years, summarization researches began to expand to software artifacts such as user stories [8], source code [9] and bug reports. For bug reports, sentence-level extractive model is the main summarization technique, which extracts the central sentences from the original text in accordance with a certain compression ratio.

Rastka et al. [10] first proposed a model to extract summaries automatically from bug reports and created a bug report corpus called BRC corpus. They extracted more than 20 features from each sentence and trained Logistic Regression models to select sentences from a bug report. He Jiang et al. [11] found the relationship between the writing style consistency and the quality of the written report by mining authorship characteristics in bug repositories. And they proposed a new summarization method named Authorship Characteristics based Summarization (ACS). In addition, they also proposed a PageRank-based Summarization Technique (PRST) [12], which utilizes the textual information contained in bug reports and additional information in associated duplicate bug reports. Lotufo et al. [13] proposed an unsupervised bug report summarization approach that estimates the attention a user would hypothetically give to different sentences in a bug report when pressed with time. In addition, Yeasmin et al. [14], [15] applied the Lotufo's model to practice and constructed a visualized summarization model.

Previous researches have yielded good results by mining different features of bug reports. Moreover, intentions of sentences have been used in many other software artifacts except for bug reports, providing us with a new perspective to explore the characteristics of bug reports. Sorbo et al. [4] proposed a novel, semi-supervised approach named DECA (Development Emails Content Analyzer) that used Natural Language Parsing to classify the sentences in development emails according to their purpose. They said that their work can be used in summarizing work and they have already generated app review summary based on intention classification [3].

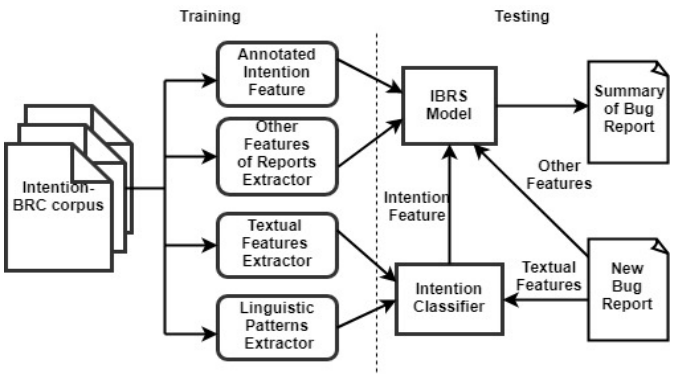

Fig. 2. IBRS Approach Overview

III. APPROACH

This section introduces the framework of IBRS as shown in Figure 2. It consists of features extractor, intention classifier and Intention-based Bug Report Summarization Model. Our goals are to build a taxonomy for intentions in bug reports, construct an automatic intention classifier to get sentences intentions and take advantage of the sentences intentions to improve the original bug report summarization approach.

A. Intention Taxonomy

In the work of Sorbo et al. [4], they divided the intention of development email sentences to 6 categories, including feature request, opinion asking, problem discovery, solution proposal, information seeking and information giving. Through observation, we find that many of their intention categories are similar to intentions in bug reports. For example, problem discovery in development emails is closely related to bug description in bug reports. Solution proposal in development emails is closely related to fix method in bug reports. However, bug reporters mainly focus on the discussion of bugs. Through reading bug reports and referring to the intentions of development emails, we define a total of 7 different intentions for bug reports as shown below.

- **Bug Description.** To describe a bug (i.e. What is the problem and how does it occur). E.g.*"The problem here is that nsSearch-Suggestions.js is passing the wrong previous Result to form history."*
- **Fix Solution.** To describe how to fix a bug. E.g. *"The simple fix it to just discard the service's form history result copy when startSearch() is called with a null previous result."*
- **Opinion Expressed.** To Express the developers' ideas. E.g. *"Yes, agreed the pref is not ideal for this purpose."*
- **Information Seeking.** Developers ask for information. E.g. *"Can I commit gtkshow.c to gtk+ when I remove gtk_show_help?"*
- **Information Giving.** Developers give their suggestions or other information. e.g. *"You could winding kdebase/apps/konsole back a few revisions to see if the problem disappears."*

- **Meta/Code.** A sentence that mainly consists of code, stack information and other meta data. E.g. *"CreateFileOperation op1 = new CreateFileOperation(file,...);"*
- **Emotion Expressed.** Greeting words or feeling expressed of something. E.g. *"Good point." "Hi Martin"*

B. Intention-BRC Corpus

In order to mine the intentions in bug reports, we annotate the intention of each sentence in the BRC corpus. BRC corpus is created by Rastka et al [1]. There are 36 bug reports (2360 sentences) in it, which from 4 different open-source software projects: Eclipse Platform, Gnome, Mozilla and KDE. They assigned three annotators to each bug report to select sentences that should occur in the summary of this report. For each bug report, the set of sentences which be marked as summary by more than one annotators is called the gold standard summary (GSS). They also roughly classified each sentence and labelled the categories. However, the categories annotations in the corpus are not very accurate and some of the categories annotations are missing. We re-annotate each sentence in the corpus according to the intention taxonomy we defined. Moreover, we correct some incorrect ID of sentences in BRC corpus. After that, we call the modified corpus as Intention-BRC Corpus in this paper and make it public[1].

C. Intention Mining

The category information of intentions can be used as a distinctive feature in bug reports. To prove this, in this section, we analyse the Intention-BRC Corpus.

The annotation result shows that there are 374 sentences labelled as *Bug Description*, 164 sentences labelled as *Fix Solution*, 239 sentences labelled as *Opinion Expressed*, 85 sentences labelled as *Information Seeking*, 588 sentences labelled as *Information Giving*, 744 sentences labelled as *Meta/code*, 166 sentences labelled as *Emotion expressed*. We wonder whether different categories of intentions are well-differentiated from other features of a bug report. So we analyse the following data:

The probability of a sentence that appears in the summary. In fact, when people reading a bug report, they always pay different attention to sentences of different intentions. For example, *"Good point"* and *"The simple fix it to just discard the service's form history result copy when startSearch() is called with a null previous result"* are sentences with different intentions. Obviously, the last sentence is more likely to appear in summary. We counted the number of sentences with each intention that appeared in summary in Intention-BRC Corpus as c_1, the number of sentences that not appeared in the summary as c_2. Figure 3 shows the result. The probabilities for sentences with *Bug Description* and *Fix solution* intentions to appear in summary are significantly higher than the others.

We define parameters P_i as the probability to be selected as summary for intention category i. In our definition, P_i is proportional to the ratio of the number of occurrences in the summary and the total number of sentences of intention i. Formally,

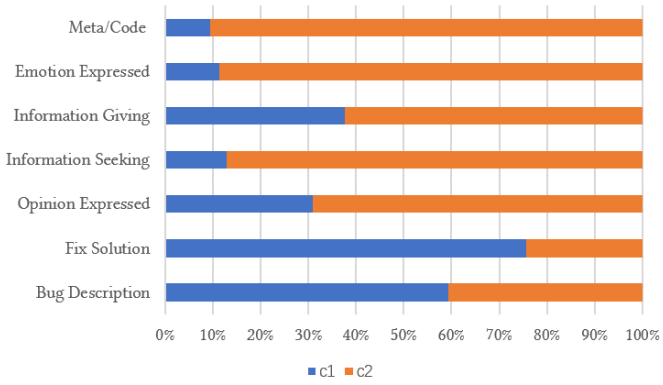

Fig. 3. Sentence Count for Each Intention

$$P_i = \frac{\sum_{s \in sen_i} \mathbb{I}(s \in sumary)}{\sum \mathbb{I}(s \in sen_i)} = \frac{c_1}{c_1 + c_2} \tag{1}$$

sen_i represents the set of sentences with intention i.

Sentences length of each intention. In general, the length of sentences with different intentions are different due to the amount of information and information types involved are different. For example, the *Bug Description* is to describe a problem (e.g. how the problem occurs) and the sentences of this intention is usually very long. We calculate the average sentence length for each category, formally,

$$L_i = \frac{\sum length(s \in \ sen_i)}{\sum \mathbb{I}(s \in sen_i)} \tag{2}$$

Sentences position of each intention. We observe that there is also a big difference in the location distribution of sentences with different intentions. For example, problems are usually described first in a bug report, therefore *Bug Description* sentences usually appear at first comment. Similarly, we calculate the average location of sentences for each category, formally,

$$C_i = \frac{\sum comment\ position\ in\ report(s \in \ sen_i)}{\sum \mathbb{I}(s \in sen_i)} \tag{3}$$

$$S_i = \frac{\sum position\ in\ comment(s \in \ sen_i)}{\sum \mathbb{I}(s \in sen_i)} \tag{4}$$

Table I shows the calculation result of PLCS for each intentions. Through the above data analysis, we can see that the categories of intentions can show differences in multiple feature dimensions. It shows that the intention implicitly expresses several features of bug reports to some extent.

D. Intention Classification

In order to automatically obtain the intention of sentences in bug reports, we train an intention classifier. Sorbo et al. [4] proposed an approach called DECA to classify the

[1]https://github.com/HuaiBeibei/IBRS-Corpus

TABLE I
PLCS FOR EACH INTENTION

Intention category	P	L	C	S
Bug Description	0.594	16.660	3.786	5.805
Fix Solution	0.756	17.226	8.726	4.006
Opinion Expressed	0.310	16.272	8.732	3.967
Information Seeking	0.129	12.388	8.612	3.967
Information Giving	0.378	17.252	7.932	6.129
Meta/Code	0.114	9.212	6.056	23.659
Emotion expressed	0.095	4.723	6.837	5.753

sentences in development emails. They created 231[2] heuristics to detect common linguistic patterns in sentence to predict their intention. However, finding linguistic patterns is a tedious work and it is impossible to define all patterns. Therefore, we create a mixed model which consists of linguistic patterns matching and machine learning classifying. For each sentence to classify, we first try to find the linguistic patterns matching it to get the intention category. If no patterns matched we input it to the trained machine learning classifier to get the intention category.

Machine learning classifier. First, we pre-process the textual content by applying stop-word removing, stemming and lowering. Then we use bag-of-words model to construct a Term-by-Documents Matrix M where $M_{i,j}$ represents the weight of the *i-th* term contained in the *j-th* sentence. We weight word using the *tf* (term frequency), which weight each word *i* in document *j* as:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{5}$$

where $n_{i,j}$ is the frequency of word *i* appear in document *j*. We use *tf* instead of *tf-idf* indexing for the use of the inverse document frequency (*idf*) penalizes too much on terms appearing many times in documents. In our work, we need these frequent words (such as "bug", "think") guide our machine learning classifier.

Linguistic patterns. We define several regular expressions and reuse part of the 231 heuristics (Some of the heuristics can also apply to our intention taxonomy, e.g. problem discovery could be regard as bug description, solution proposal could be regard as fix solution). Following are some examples to identify intentions using linguistic patterns. Each example consists of a linguistic pattern and a matching sentence from bug reports.

- Example 1: Identify Code Category.
  - \w+\s\w+\s\(.+\)
  - *"int ftp_connection (FtpConnection *conn, const Ftp550Handler *handlers, const FtpFile *file)".*
- Example 2: Identify Information Seeking Category.
  - $[how|what|left|Do|Does|Is].+\?$
  - *"**Does** this happen every time**?**"*
- Example 3: Identify Information Giving Category.
  - *[someone] could [verb]*
  - *"**You could winding** kdebase/apps/konsole back a few revisions to see if the problem disappears"*

[2]http://www.ifi.uzh.ch/seal/people/panichella/DECA_Implemented_Heuristics.pdf

E. Improved Automatic Summarization of Bug Reports

Different from the previous methods, we combine the intention feature to the summarization model. The score of sentence *s* can be expressed by the following formula:

$$F_s = (1-\alpha) * BRS_s + \alpha * P_{intention(s)} \tag{6}$$

BRS_s is the score output by the original BRC model. $intention(s)$ is the intention category of sentence s and $P_{intention(s)}$ is the probability weight for this intention category. In fact, we can add the intention feature to machine learning model to learn the α. Following are the features we used to train IBRS model:

1) *structural features* are related to the structure of the bug report (e.g., the position of the sentences).
2) *participant features* are directly related to the conversation participants (e.g., whether the sentence is made by the same person who filed the bug report).
3) *length features* related to the length of the sentence
4) *lexical features* are related to the occurrence of unique words in the sentence that we could use to calculate the sentence similarity with bug report title.
5) *Intention features* are related to the weight of intention category of the sentences.

IV. EXPERIMENTS

A. Research Questions

In this paper, we are interested in the following research questions and conduct two experiments to evaluate our approach:

RQ1: How does our intention classifier perform? For this question, we construct the intention classifier and evaluate it on Intention-BRC corpus.

RQ2: Does the sentence intention feature improve the result of bug report summary model? To answer this question, we add the intention feature to the original summary model to construct IBRS and evaluate it in experiment II.

RQ3: What is the impact of missclassification to IBRS? To answer this question, we use labelled intentions in corpus rather than the predicted intentions of the intention classifier to construct IBRS and evaluate it in experiment II.

B. Experiment I on Intention classifier

Algorithm 1 show the main algorithm of our intention classification approach. We choose Random Forest (RF) [16] classifier and implement leave-on-out method on RF to make sure the sentence to classify not appear in the train set.

To evaluate the result of the intention classification, For each category, we calculate the precision, recall and F-score. The result of the classifier is shown in Table II.

Answer for RQ1: Through experimental verification, we can correctly identify the intention of 59% of the sentences from

Algorithm 1 Intention Classification

```
Split the BRC corpus by sentences
for each sentence s ∈ BRC corpus do
  for each linguistic patterns defined do
    if the sentence i match the pattern then
      return the intention the pattern represent
    end if
  end for
  return the sentence predicted by the trained RF classification
end for
```

TABLE II
EVALUATION MEASURES OF EXPERIMENT I

CLASS	TP	FP	FN	Precision	Recall	F-score
Bug Description	182	186	192	0.49	0.47	0.48
Fix Solution	19	63	145	0.23	0.12	0.16
Opinion Expressed	27	39	212	0.41	0.11	0.17
Information Seeking	60	36	25	0.63	0.71	0.67
Information Giving	364	375	224	0.49	0.62	0.55
Meta/Code	679	251	65	0.73	0.92	0.81
Emotion Expressed	52	27	114	0.66	0.31	0.42
Overall Performance	1383	977	977	0.59	0.59	0.59

bug reports. Especially, our intention classification approach performs better at several categories, for example, *Meta/Code*, *Information Seeking*, *Information Giving*. This is because the sentences with these intentions have more obvious structural features. For example, code sentences' keywords have a strong distinction, such as *"public"*, *"static"*, etc. *Information Seeking* sentences are often in the form of questions and usually contain obvious keywords such as *"what"*, *"how"*, *"why"*, etc. The worse performing results are from *Fix Solution* and *Opinion Expressed*. There are two main reasons. One reason is that the count of these two category samples is much smaller than the others (7% *Fix solution* sentences and 10% *Opinion Expressed* sentences). The other reason is that the diversity of sentence structure of these two categories. In addition, since bug report repository is a relatively open platform, developers do not write a standardized language when reporting (for example, they usually use abbreviated form).

C. Experiment II on IBRS

This experiment combines the intention classifier and BRC model. Similar to the method in experiment I, we also use a leave-one-out procedure (i.e. leave one bug report out). The process of the IBRS experiment is as following:

1) Leave one bug report out of the training set. For every sentence in bug report corpus, we get their intention category using the intention classifier trained without sentences in this bug report.
2) We use the remained set in intention-BRC corpus to train the IBRS model. We map the intention label to get the sentence intention feature(P) from Table I and add the intention features so that our new summary model can take the category of intention into account.

TABLE III
EVALUATION MEASURES OF EXPERIMENT II

Approach	Precision	Recall	F-score	Pyramid Precision
BRC	0.57	0.35	0.43	0.66
BRC*	0.54	0.34	0.42	0.64
IBRS	0.59	0.37	0.45	0.69
IBRS*	0.65	0.41	0.50	0.72

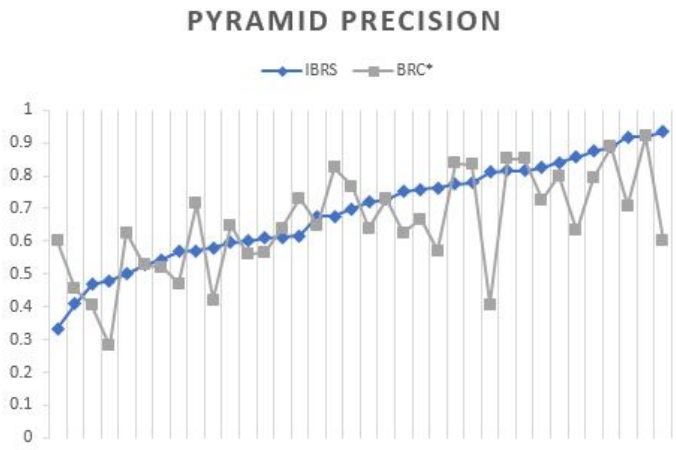

Fig. 4. Pyramid Precision For IBRS and BRC*

3) For the left bug report, we map the sentences to get the sentence intention feature. Then use the IBRS to get the sentence score. We sort the sentences by their scores and select top 20% (for that 20% is approximately equal to the proportion of GSS in the corpus) as summary.

To evaluate the result, we calculate the precision, recall, F-score and pyramid precision [1] for the IBRS model. In order to compare with original models, we also reproduce the BRC experiment proposed by Rastkar et al. To answer RQ3, we also change the experiment to use the labelled intentions rather than the predicted intentions at step 1). The result is shown in Table III.

In our reproduced experiment, we get precision of 54% and recall of 34% (We correct several errors in the corpus of BRC annotations that may have caused a slight difference between the reproduced results and the experimental results of Rastkar et al.) as showed in second row in Table III marked as BRC*. The original result of BRC approach is shown in first row. The result of IBRS is shown is third row. IBRS* shown in last row is the result of IBRS model trained using correctly labelled intentions.

We also calculate pyramid precision for every single bug report. Figure 4 shows the values of pyramid precision for the BRC* and IBRS approach. The bug reports have been sorted based on the pyramid precision of IBRS. The figure shows that 22 of 36 reports (61% of total reports) get better summary using IBRS.

Answer for RQ2: Our IBRS out-performs the BRC* approach on precision (5% improved), recall (3% improved), F-score (3% improved) and pyramid precision (5% improved).

61% of bug reports get summaries with better pyramid precision using IBRS. The only different between the BRC* and IBRS is that IBRS takes the sentence intention category feature into account. It proves that the intention feature indeed improves the work of automatic summarization for bug reports.

Answer for RQ3: From the result of IBRS*, it shows that IBRS* outperforms the IBRS 6% on precision. IBRS* is trained with labelled intention while IBRS trained with the intention classifier of 59% precision. Although IBRS improves the summarization for bug reports, the misclassification limits the promotion of summarization to some extent. It proves that intentions can enhance the summarization work better. Improving the accuracy of the intention classifier is one of our future work.

V. DISCUSSION

A. Threats to Validity

In this paper, we use Intention-BRC corpus that consists of 36 bug reports. This may bring threats to the validity of intention taxonomy for the corpus are not larger enough. However, the 36 bug reports are from 4 different popular project and can represent the characteristics of most of bug reports. The size of the corpus also bring threats to the evaluation of the experiment. We apply the leave-one-out method to ensure that the assessed samples do not participate in training and make maximum use of corpus. In addition, we select 20% sentences as summary of a bug report may bring threats to the quality of the summary content. The state-of-the-art researches usually count the average proportion of sentences in summary to total sentences as an indicator of the number of sentences to extracted. 20% is the proportion of GSS to total sentences in corpus, so it is a relatively reasonable extraction ratio.

B. Future Work

There is a lot of room to improve on intention classifier. In our future work, we are interested in improving the accuracy of the intention classifier by adding more heuristics and trying new ways to characterize bug report text (e.g. DBRNN-A [17] instead of our bag-of-words (BOW)).

Moreover, the intentions of sentences can be used in many other ways. For example, we can rearrange or even reconstruct the summary text of the bug report based on the intention of the sentence. There are many redundant sentences with similar meaning in the summary text, we can further abstract the sentences with same intention into a abstract text. [18].

VI. CONCLUSION

In this paper, We achieve significant improvement to the automatic summarization for bug report by introducing intention feature to original approach. We assume that bug report text usually contains different intentions. We introduce the intention taxonomy and propose **IBRS** (Intention-based Bug Report Summarization). To evaluate the performance of the intention classifier and IBRS model, we design two experiments. The result shows the precision rate of intention classifier is 59%. Based on the intention classifier, we implement IBRS model, which outperforms the original BRC model with precision of 59% (5% improved), recall of 37% (3% improved) , f-score of 45% (3% improved)and pyramid precision of 69% (5% improved). The results of experiments show that mining intentions indeed improves the bug reports summarization.

REFERENCES

[1] S. Rastkar, G. C. Murphy, and G. Murray, "Automatic summarization of bug reports," *IEEE Transactions on Software Engineering*, vol. 40, no. 4, pp. 366–380, 2014.

[2] S. Mani, R. Catherine, V. S. Sinha, and A. Dubey, "Ausum: approach for unsupervised bug report summarization," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 11.

[3] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in my app? summarizing app reviews for recommending software changes," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 499–510.

[4] A. Di Sorbo, S. Panichella, C. A. Visaggio, M. Di Penta, G. Canfora, and H. C. Gall, "Development emails content analyzer: Intention mining in developer discussions (t)," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015, pp. 12–23.

[5] G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," *Journal of Artificial Intelligence Research*, vol. 22, pp. 457–479, 2004.

[6] R. Nallapati, B. Zhou, C. Gulcehre, B. Xiang *et al.*, "Abstractive text summarization using sequence-to-sequence rnns and beyond," *arXiv preprint arXiv:1602.06023*, 2016.

[7] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," *arXiv preprint arXiv:1509.00685*, 2015.

[8] R. Krasniqi, S. Jiang, and C. Mcmillan, "Tracelab components for generating extractive summaries of user stories," in *IEEE International Conference on Software Maintenance and Evolution*, 2017, pp. 658–658.

[9] P. W. Mcburney and C. Mcmillan, "Automatic source code summarization of context for java methods," *IEEE Transactions on Software Engineering*, vol. 42, no. 2, pp. 103–119, 2016.

[10] S. Rastkar, G. C. Murphy, and G. Murray, "Summarizing software artifacts: a case study of bug reports," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 2010, pp. 505–514.

[11] H. Jiang, J. Zhang, H. Ma, N. Nazar, and Z. Ren, "Mining authorship characteristics in bug repositories," *Science China Information Sciences*, vol. 60, no. 1, p. 012107, 2017.

[12] H. Jiang, N. Nazar, J. Zhang, T. Zhang, and Z. Ren, "Prst: A pagerank-based summarization technique for summarizing bug reports with duplicates," *International Journal of Software Engineering and Knowledge Engineering*, vol. 27, no. 06, pp. 869–896, 2017.

[13] R. Lotufo, Z. Malik, and K. Czarnecki, "Modelling the 'hurried' bug report reading process to summarize bug reports," *Empirical Software Engineering*, vol. 20, no. 2, pp. 516–548, 2015.

[14] S. Yeasmin, C. K. Roy, and K. A. Schneider, "Interactive visualization of bug reports using topic evolution and extractive summaries," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 2014, pp. 421–425.

[15] ——, "How should we read and analyze bug reports: an interactive visualization using extractive summaries and topic evolution," in *Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., 2015, pp. 171–180.

[16] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[17] S. Mani, A. Sankaran, and R. Aralikatte, "Deeptriage: Exploring the effectiveness of deep learning for bug triaging," *arXiv preprint arXiv:1801.01275*, 2018.

[18] G. Murray, G. Carenini, and R. Ng, "Generating and validating abstracts of meeting conversations: a user study," in *Proceedings of the 6th International Natural Language Generation Conference*. Association for Computational Linguistics, 2010, pp. 105–113.