AUSUM: Approach for Unsupervised Bug Report SUMmarization

Senthil Mani, Rose Catherine, Vibha Singhal Sinha, Avinava Dubey*
IBM Research - India
{sentmani, rosecatherinek, vibha.sinha}@in.ibm.com, avinava.dubey@gmail.com

ABSTRACT

In most software projects, resolved bugs are archived for future reference. These bug reports contain valuable information on the reported problem, investigation and resolution. When bug triaging, developers look for how similar problems were resolved in the past. Search over bug repository gives the developer a set of recommended bugs to look into. However, the developer still needs to manually peruse the contents of the recommended bugs which might vary in size from a couple of lines to thousands. Automatic summarization of bug reports is one way to reduce the amount of data a developer might need to go through. Prior work has presented learning based approaches for bug summarization. These approaches have the disadvantage of requiring large training set and being biased towards the data on which the model was learnt. In fact, maximum efficacy was reported when the model was trained and tested on bug reports from the same project. In this paper, we present the results of applying four unsupervised summarization techniques for bug summarization. Industrial bug reports typically contain a large amount of noise—email dump, chat transcripts, core-dump—useless sentences from the perspective of summarization. These derail the unsupervised approaches, which are optimized to work on more well-formed documents. We present an approach for noise reduction, which helps to improve the precision of summarization over the base technique (4% to 24% across subjects and base techniques). Importantly, by applying noise reduction, two of the unsupervised techniques became scalable for large sized bug reports.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Debugging aids*; I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*Text analysis*

Keywords

Unsupervised, Summarization, Bug Report

*This work was done when he was a researcher at IBM Research - India. His current affiliation is LTI, Carnegie Mellon University.

SIGSOFT'12/FSE-20, November 11–16, 2012, Cary, North Carolina, USA.

1. INTRODUCTION

Bug management systems are an integral part of any software project. They are used by the project team to record problems raised by end users and to track the investigation and resolution of bugs. As part of root cause analysis, significant amount of communication might happen between the reporter of the bug and the developers through the bug report. As a result, bug reports often resemble recorded conversations—messages from multiple people ordered sequentially where each message is composed of multiple paragraphs of unstructured text. As the bug gets resolved over a period of time, the bug report accumulates useful information about the problem reported such as the investigation and resolution details. These resolved bugs are then archived for future reference. When a new bug is reported, developers first check the archived bugs to see if similar problem was resolved in the past [11]. Finding a similar bug could help them in multiple ways: (1) in better understanding of current problem, (2) if the resolution has been recorded then, reusing it for the new problem. Especially in industrial projects, a large number of bugs might not require code changes. The reported problem could be because of some configuration or usage issue. In such cases the archived bugs become all the more indispensable [21]. This is because, the support team can quickly check if similar problems reported in past required a code fix or were solved through other means. In most cases this either reduces the time taken to solve the issue or enables a developer with relatively less experience on the project to accurately respond to the issue.

Search [11] can help a developer identify similar bugs from the archived repository. However, the developer still needs to follow the tedious process of going over each of the recommended bug and identify if the bug report contains any information of interest or not. Wading through complete contents for each recommended bug might be too time consuming and frustrating. For example, the bug reports in our two experiment subjects on an average had 66 and 332 sentences respectively. In [34], the authors suggest that one way to reduce the time a practitioner takes in identifying the correct bug report is to provide a summary of each recommended bug. The ideal way to do this would be that after a bug is resolved, an assigned developer manually writes out an abstract. Figure 1 shows an example of the bug report and it's corresponding summary. This abstract could further be used by other developers in the system, to get a better understanding of the bug. However, this ideal method is unlikely to work in practice, since it involves huge manual effort. Hence, there is a need for automatic summarization of bug reports.

There are two approaches available for summarizing textual artifacts. One is a learning based supervised approach—various variants of which are described in [26, 43, 45, 18]. The essence of

Comment # 1 :
Hello Advanced Support.

I have a customer who is migrating his DB2 instance to 64-bit and will keep his application 32-bit. How can we avoid data truncation issue faced by the customer?

Hello
executed the following script :
Proc1 (CURRENT TIMESTAMP2) -3 sqlexecdirect 1 drop procedure regress.

Comment # 2 :
To avoid data truncation it is required he change all variables defined as type ` long ' to be defined type ` sqlint32 ' or ` sqluint32 ' .

Comment # 3 :
The customer would like to avoid manually making these changes to all the source code files in their application and wonders if he can use the preprocessor option of the PRECOMPILE (1) command to make the changes at precompile time . If yes could you provide an example of the syntax that would be needed . I appreciate your continued support in helping with this issue.

Sample Bug Report

I have a customer who is migrating his DB2 instance to 64-bit and will keep his application 32-bit.

To avoid data truncation it is required he change all variables defined as type ` long ' to be defined type ` sqlint32 ' or ` sqluint32 ' .

The customer would like to avoid manually making these changes to all the source code files in their application and wonders if he can use the preprocessor option of the PRECOMPILE (1) command to make the changes at precompile time .

Bug Report Summary (30% of sentences)

Figure 1: Example of Bug Reports and it's Summary

these approaches is as follows: (1) ask users to manually summarize a set of documents (training set), (2) extract out a set of text features from these documents and train a statistical model. The model tries to identify the features that help predict the manual summaries provided in the training set, (3) for a new document, extract the features of interest from the document and use the trained model to predict it's summary. The other is an unsupervised approach—various variants presented in [24, 13, 46, 47]. These assign centrality and diversity measures to various sentences in a document and use these measures to select the sentences to be put in the summary. We describe these measures later in the paper in Section 2.2.

In a prior work [34], Rastkar et al. applied supervised learning approaches to summarize bug reports. Supervised learning approaches have their drawbacks as they involve collecting manually labelled training data and once a model is trained, the efficacy of the model depends highly on the similarity of the training and actual data on which it is applied. The practical application of such a supervised technique in any project could be hampered owing to the initial training cost involved. The authors reported a pyramid precision([8]) of 63% when the model was trained on bug reports from same subject against 54% precision when the model was trained on a corpus of mail threads.

In our work, we applied four unsupervised approaches *Centroid* [32], *Maximum Marginal Relevance (MMR)*[5], *Grasshopper*[48] and *Diverse Rank*[23] to the problem of summarization of bug reports on the dataset used in [34][1](SDS) and one internal industrial project dataset (DB2-Bind). Across the two subjects, *DivRank* and *Grasshopper* had convergence problems on 9 large bug reports which resulted in no summaries being generated for them. For the remaining bug reports in *SDS* dataset, when compared to supervised techniques, all the unsupervised approaches provided better recall (an improvement by 16%). There was however a 3% dip in pyramid precision between the best of unsupervised approach *MMR, GrassHopper* and the best of supervised approach i.e when the approach was trained on bug reports from the same subject. But, all unsupervised approaches provided comparable precision to the supervised approaches that were trained on a different dataset other than the subject where they were applied.

The drop in precision and the convergence issues made us believe that bug reports have specific type of data that is causing the generic unsupervised approaches to not perform as well as the supervised approaches. Bug reports resemble conversations, very often with email and chat content pasted. This content contains email headers, salutations like "hello","hi" and closing phrases like "thank you", "thanks" etc that are not useful from summary perspective. There are very often long stack traces, command outputs pasted that are not that useful from summarization perspective. To handle this, we augmented our summarization approach by adding a pre-processing step to filter out such sentences (noise) from the bug reports. This noise removal step helped improve the precision of the base unsupervised algorithms. Also, the algorithms *DivRank* and *Grasshopper* which failed to converge and produce any summaries for few large bug reports, successfully generated summaries for the same bug reports once the noise was filtered from them using our approach. The main contributions of the paper are as follows:

- A classification scheme for bug report content and a novel noise reducer to classify bug report sentences into this classification automatically. This bridges the gap between theory and practice.
- Experimental evaluation of four well known unsupervised summarization algorithms to the problem of summarizing bug reports. The results indicate that unsupervised approach generates as good a summary as supervised approach proposed in earlier paper.
- The summarization approach was enhanced to filter out specific classes of content from bug report that could be perceived as noise from summarization perspective. Experimental validation of the impact of using a noise filter before applying the unsupervised techniques indicates that better summaries are generated.

Rest of the paper is organized as follows. In section 2 we describe our approach. In Section 3, we give a brief overview of the unsupervised summarization algorithms we used. Section 4 describes the experimental studies we conducted. We present some observations and threats to validity in Section 5. The related work is presented in Section 6 and conclusions in Section 7.

[1]Available at: http://www.cs.ubc.ca/labs/spl/projects/summarization.html

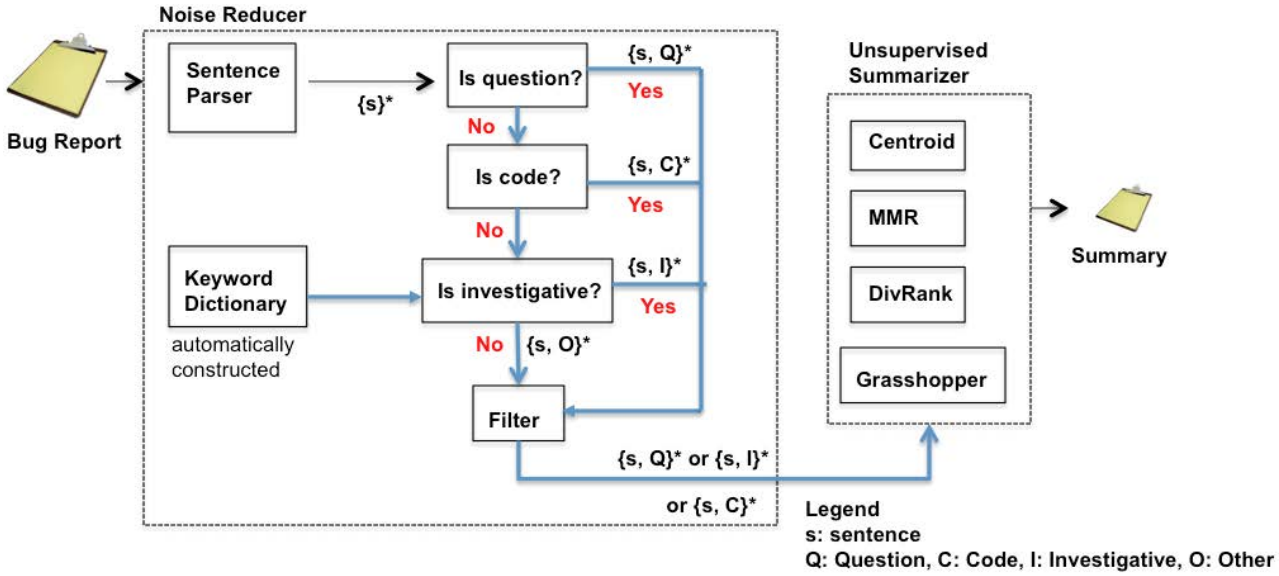

Figure 2: Approach for Unsupervised Bug Report Summarization

2. APPROACH

Figure 2 outlines our summarization approach. For a bug that needs to be summarized, we first pass it through the noise reducer module. Here, we broadly classify each sentence into *question*, *investigative* sentence, *code* fragment and *others*. In different variants of our approach, we filter out different type of sentences and pass the filtered set to a summarizer, which applies the unsupervised techniques to extract the summary from the set of "useful" sentences. For example in one variant, the *other* sentences are filtered out and in another, both *code* and *other* sentences are removed. The filtered content is passed to the summarizer module. This modules has 4 different techniques for summarization plugged in: Centroid based, MMR, DivRank and Grasshopper. In the subsequent subsections we describe the noise reducer and summarizer module in detail.

2.1 Noise Reducer

We analyzed the nature of content in bug reports and came up with following classification for information in bug reports.

- **Question Sentences**: these sentences describe the problem being reported and are usually contain words such as *why, how, what*.
- **Investigative Sentences**: these sentences describe some options that the user can try to further investigate the issue or give more details on understanding the problem. These sentences would typically contain application specific information often indicated by presence of domain specific keywords.
- **Code Sentences**: these sentences contain code fragments, stack traces or command outputs that the user might have provided as part of initial bug description, or as part of investigation or solution.
- **Others**: These are very often greeting sentences, for example "Hello", "thank you for your support".

Figure 3 shows the sentence classification for the sample bug report shown in Figure 1. The bug report consists of series of communication as comments. Each comment in turn contains a set of sentences. In Figure 3, sentence S3 is classified as *Question*. Sentence S5 in an example of a *Code* sentence. Sentences S2, S6, S7 contain key words like 64-bit, 32-bit, sqlint32, sqluint32, preprocessor, PRECOMPILE and are examples of *Investigative* sentences. Sentences S1, S8 and S9 are examples of *Other* category.

As part of our initial investigation, we had summaries created manually by 5 different developers. We further analyzed the sentences used in these summaries and found that all these sentences were either *Question* and *Investigative* type. Based on this we concluded that if we were to filter out all sentences from bug reports which were not of these kinds and then run the summarizer, we would not lose any useful information, but rather improve the efficacy of the summarization.

In order to implement a system that can classify bug report content into the above mentioned classes, we had two design choices. First, follow a supervised approach, where we manually classify (label) some percentage of the sentences into the four categories, train a statistical model and then execute the model on remaining set to predict the classification.However, this approach would have the disadvantage that the model needs to be re-trained for new subjects. Second option was to follow an unsupervised approach. Leverage the sentence structure and develop heuristics to come up with a classification. We chose to follow the latter approach, as it can be widely applied for different subjects (datasets), and the effort involved in fine-tuning the heuristics will be way less than manually classifying (labeling) for new datasets.

The noise reducer works as follows. First, we convert the bug report into a sequence of sentences. For this we use an off the shelf english text parser, Stanford NLP parser[2]. Once we have the sentences we further classify each sentence into one of the four categories using heuristics mentioned below. Finally, we filter out the sentences not useful from summarization perspective, and pass this filtered set to the summarizer.

2.1.1 Question Sentences

A sentence can be marked as question, if it starts with words such as "what/why/where" etc and optionally ends with a question mark. Regular expressions as follows could be used to identify questions.

[2]http://nlp.stanford.edu/software/lex-parser.shtml

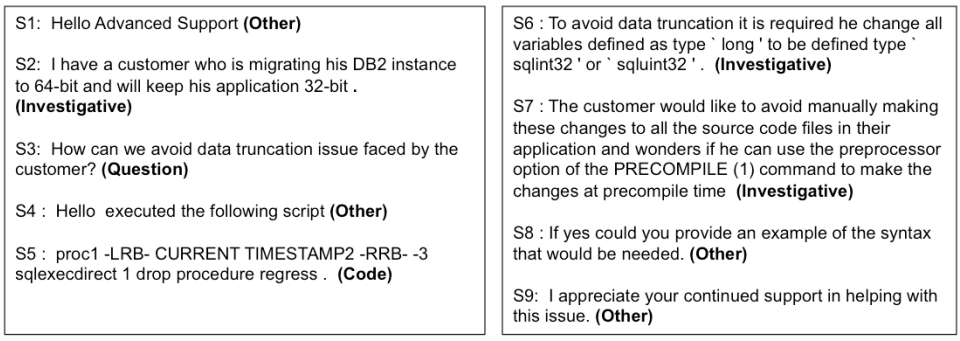

Figure 3: Example: Sentence Classification of Sample Bug Report

```
^[Ww]hat.*?*$
^[Hh]ow.*?*$
```

Another, way to identify questions is to use the parsed syntactic structure of sentences. Sentences that contain node SBARQ or SQ[3]in their parse tree, are typically questions. For example, the parsed structure of S3 in Figure 3 is as follows:

```
(ROOT
  (SBARQ
    (WHADVP (WRB How))
    (SQ (MD can)
      (NP (PRP we))
      (VP (VB avoid)
        (NP
          (NP (NN data) ..)
          (VP (VBN faced)
            (PP .. )))))
    (. .))))}
```

We used the parsed syntactic structures to classify sentences as *Question* type.

2.1.2 Code Sentences

To classify sentences into code, we analyzed the bug report contents and came up with patterns that indicate presence of stack trace, java code, commands, command output. Some examples of these patterns are:

```
starts with : "db2","proc", "publicÓ
contains :  "<", "SQL", "{", "}",
            "else if.*(*", "if.*(*",
            " public static",
            " int ", " char Ó
ends with : ";"
```

If a sentence satisfies any of the above rules, it is classified as a code sentence. While these patterns are fairly generic, they might have to be enhanced only in special cases.

2.1.3 Investigative and Other Sentences

The non-question, non-code sentences can be further classified into *Investigative* and *Other*. Investigative sentences give insight into the problem and/or solution being proposed or implemented. These typically contain some application specific information, as indicated by presence of 64-bit, 32-bit in sentence S2 in Figure 3. Any sentence that does not contain application specific information, could potentially be marked as others. To classify sentences into these two categories, we applied the following heuristic: if a sentence was of a minimum length of 5 words and contained more than two application specific keywords, it could be marked as *Investigative*. We derived the keyword dictionary from the bug report content itself as explained in the next subsection.

[3]Penn Treebank Tags http://www.cis.upenn.edu/ treebank/

2.1.4 Keyword Dictionary

Intuitively, what we are trying to achieve here is to identify a ranked order of words in each bug report, that indicates the discriminating power of that word in the bug report. This relative ranking is obtained by leveraging the term frequency-inverse document frequency metric [22] typically used in text search.

To create the dictionary, we first extract the terms (i.e. single words) from all the bug reports available for a subject. For each term (t) per bug report (d), we calculate it's *tf-idf(t,d)* (term frequency-inverse document frequency). Term frequency *tf(t,d)* is defined as the number of instances of a term in a document normalized by the document size.

$$tf(t,d) = \frac{\sum t}{|d|} \tag{1}$$

The inverse document frequency *idf(t)*, for a term (t) is the measure of general importance of the term across the subject. It is obtained by dividing the total number of bug reports (D) by the number of reports containing the term (d) , and then applying the logarithm.

$$idf(t,D) = log\frac{|D|}{|d \in D : t \in d|} \tag{2}$$

Then the *tf-idf* is calculated as

$$\textit{tf-idf(t,d)} = tf(t,d) * idf(t) \tag{3}$$

For each term per bug report, we further normalize the *tf-idf(t,d)* w.r.t. the highest *tf-idf* obtained for that report. The key words in the document were then picked up by applying a cut-off on the normalized *tf-idf*.

2.1.5 Filter

For each bug report, once the sentences are classified into these four categories, then we can selectively choose different classes of sentences or combinations thereof to use as inputs for summarization. In our experiments, we have evaluated the impact of: (1) removal of *Other* sentences and (2) removal of *Other* and *Code* sentences.

2.2 Unsupervised Summarization

Unsupervised summarization refers to the method of summarizing a text or document without using a trained model. This has the obvious advantage of not requiring any labeled data, which is usually difficult and/or costly to collect due to the manual effort involved. It is usually classified into two broad types - extractive and abstractive [15]. An extractive summarization refers to any summarization method in which the sentences in the summary are chosen from the input document and are used as is. In abstractive summarization, the sentences of the summary are usually different from

what is present in the original text and could involve paraphrasing. Majority of automatic summarization techniques developed so far are extractive in nature. Though abstractive summaries are closer to what a human would have constructed, the complex nature of the methods and lower returns in terms of accuracy has greatly reduced their applicability in various domains [15].

In extractive summarization techniques, an algorithm decides at the specified granularity level, whether the piece of text in the input document should be present in the output summary or not. The granularity can be at the level of a phrase, sentence, or in the case of bug reports, at a comment level. In this paper, we evaluate the summaries at the granularity level of a sentence.

Unsupervised summarization methods work by choosing sentences that are central to the input document. This centrality can be measured in various ways. A simple method for the same is to compute the 'centroid' of the document. Then, the sentences chosen for the summary are based on their distance from the centroid [30]. In this paper, we refer to this method as *Centroid*.

A more systematic method of measuring centrality is based on the amount of similarity to other sentences in the input document. This is based on the reasoning that, sentences that form the essence of the document are repeated in different ways, and thus have more overlapping terms with other sentences, than those that are not central to the document. Similarity to other sentences can be considered as a recommendation by those sentences [24]. For example, in this paper, a sentence that contains 'unsupervised summarization' and 'bug reports' would overlap with a large number of other sentences, since these phrases form the theme of this paper, thus making that sentence an important entry in the summary.

Though a centrality measure would suffice to select the important sentences of the document, it can make the sentences in the summary, repetitive. Diversity is an important notion in summarization, where the new objective is to choose sentences that are central, but at the same time, have low redundancy among themselves and fully represents or covers the document. One of the first works to propose diversity in summarization is the Maximal Marginal Relevance (MMR) algorithm [5], which is discussed in Section 3.2. MMR algorithm is also one of the most well-known works in the area of summarization, but it has been generally regarded to be lacking a principled mathematical treatment in its use of heuristics. The next important work and one of the first ones to propose a mathematical model, is the Grasshopper algorithm [48]. This method is discussed in Section 3.3. A recent work in summarization which was proposed as an improvement to the Grasshopper algorithm is the DivRank algorithm [23]. We discuss this in Section 3.4.

In this paper, we employed four extractive unsupervised summarization techniques, namely, Centroid, MMR, Grasshopper and DivRank, to summarize bug reports. In the next section we give a brief overview of each of these techniques.

3. OVERVIEW OF UNSUPERVISED ALGORITHMS USED

In this section we give a brief overview of each of the four unsupervised algorithms we used for our experiments.

3.1 Centroid

Centroid method is a simple technique for extracting summary sentences from the input document. In this, each unit of text is represented as a weighted vector of $TF \times IDF$ (Term Frequency times Inverse Document Frequency). The unit of text in our paper is a sentence from the bug report. The algorithm then proceeds by finding a centroid sentence [30, 32], which is a pseudo-sentence whose vector has a weight equal to the average of all the sentence vectors in the report. Sentences that contain words from the centroid are more indicative of the topic of the document and hence are chosen in the summary. For each sentence S_i, the algorithm defines a term called *Centroid Value* of S_i, which is calculated as the sum of the corresponding weights in the centroid sentence, of the terms in S_i. Once the centroid values of sentences have been calculated, the summary is constructed by choosing sentences in the decreasing order of their centroid values.

3.2 MMR

The Maximal Marginal Relevance (MMR) [5] based summarization proposed by Carbonell et al. suggested that, a sentence should be included in the summary if it has minimal similarity to the already chosen sentences, in addition to its centrality. This criteria is expressed as a linear combination of the centrality of the sentence and its dissimilarity to the already chosen summary sentences. Here, the dissimilarity is computed as the negative of cosine similarity, but can be the negative of any general method for computing similarity of documents. The algorithm proceeds by incrementally adding sentences to the summary, where at each step, it greedily chooses that sentence which has the maximum value for the above linear combination.

3.3 Grasshopper

Graph **R**andom-walk with **A**bsorbing **S**tate**S** that **HOP**s among **PE**aks for **R**anking (GRASSHOPPER) [48] is a graph based method proposed for ranking documents, with an emphasis on diversity. It represents the document to be summarized as a graph, where each sentence becomes a node. The edges between these nodes are assigned a weight equal to the similarity of the sentences. Now, the algorithm performs random walks on this graph similar to Google's PageRank [27] algorithm, to get the stationary distribution of the graph, which is the probability of the random walk visiting the node. Nodes with higher probability are more central to the document than others. When a random walk reaches a node that is already included in the summary, the walk is discontinued and a new node is chosen to start the walk. The next node chosen is the one that has the highest *expected number of visits*, before the walk initiated from that node would end up in an existing summary node. Here, the reasoning is that, nodes that are closer - highly similar - to the already chosen nodes, will have only few visits before being absorbed. In contrast, nodes that are farther to the chosen nodes - dissimilar to the chosen ones - still allow the random walk to linger among them, and hence will have more number of visits before eventually getting completely absorbed.

3.4 DivRank

Diverse Rank (DivRank) [23] is a recent work that also uses random walks on graph representation of data. The random walk used in this method is a *time-variant* random walk where the probability of moving from one node to another (transition probability) varies as time changes. The particular method used in [23], which belongs to the family of time-variant random walks, is called a *vertex-reinforced random walk* [28], where the basic idea is that, the transition probability to one state from others is reinforced by the number of previous visits to that state, which causes the transition probabilities to vary as time passes. Though DivRank was shown to perform better than MMR and Grasshopper, our experiments tend to prove otherwise (Section 4).

For the experiments, we used home-grown implementations of all the four algorithms. Also, it should be noted that, all these al-

Table 1: Details of the subjects used for our experiments

Subjects	No. of Bugs	Average Comment Count	Average Comments Size	Total Sentences
SDS	36	6.5	9.5	2361
DB2-Bind	19	21	16	6304

gorithms have one or two hyper-parameters that can be fine tuned if training data is available. However, we used the default values to keep it completely independent of the data. The next section presents the evaluation of our summarization approach.

4. EXPERIMENTS

We conducted three experiments to evaluate our summarization approach and answer the following research questions :

- **RQ1 - Feasibility of Unsupervised techniques for summarization** : Are unsupervised techniques applicable for generating summaries? How do they perform when compared to supervised techniques presented in earlier work [34]?
- **RQ2 - Impact of Noise identifier on effectiveness of Unsupervised techniques** : How does the efficacy of the unsupervised techniques improve for bug summarization after the noise has been filtered from the bug reports?
- **RQ3 - Goodness of Noise Identifier** : How good is the noise identifier based classification of sentences into *Question*, *Investigative*, *Code* and *Others*?

We first provide the experimental setup followed by the results of the experiments.

4.1 Experimental Setup

We conducted our experiments on two subjects. The first subject, referred to as *SDS* was obtained from [34]. This corpus was created and used by the authors to evaluate their supervised summarization technique[4]. The second subject was obtained from IBM DB2 team. We got 19 bug reports corresponding to the DB2 Bind component. This subject is referred to as *DB2-Bind*. In total, our dataset consisted of 55 bug reports.

Table 1 lists the total number of bugs, average number of comments, average comment size and total number of sentences per subject. The *SDS* corpus contains 36 bug reports from four different open-source software projects: Eclipse Platform, Gnome, Mozilla and KDE (9 from each). The reports vary in length: 25 reports (69%) had between five and fourteen comments; the remaining eleven bugs (31%) had 15 to 25 comments each. Nine of the 36 bug reports (25%) were enhancements to the target system; the other 27 (75%) were defects. There are a total of 2361 sentences in these 36 bug reports.

The *DB2-Bind* corpus contains 19 bug reports. The reports vary in length : 9 reports (47%) had between two to ten comments; the remaining nine out of 10 reports had 11 to 50 comments. Only one bug report had around 114 comments . We chose not to remove the outlier so as to evaluate the effectiveness of the unsupervised approaches on very large bug reports. Figure 4 shows the distribution of bug reports based on the number of comments. There are a total of 6304 sentences in these 19 bug reports. Even though the number of bug reports is less, it exceeds *SDS* in total sentence count.

4.1.1 Oracle Creation

To create the oracle (or ground truth) for our *DB2-Bind* subject, we asked two members from the DB2 Bind development team to manually summarize the 19 bug reports. We asked them to mark out the sentences they would want to include in the summary. Table 2 lists some basic statistics on the manual summary. We also performed the Kappa test [14] to measure the level of agreement between the two members, as typically each individual can summarize the bug report in their own ways. The κ value was 0.415, showing a moderate level of agreement. For *SDS*, the summarization was also provided along with the bug corpus. They had used three annotators per bug to mark out a summary for the 36 bug reports. Across the annotators they had a κ value of 0.41, again indicating a moderate level of agreement.

[4]http://www.cs.ubc.ca/labs/spl/projects/summarization.html

Table 2: Extractive summary manually created for DB2-Bind by two annotators

	Annotator 1		Annotator 2	
	mean	stdv	mean	stdv
#sentences in the summary	17.3	12	22.2	16.2
#words in the summary	395	296	497	407.3

4.1.2 Metrics to measure Efficacy of Summarization

We use four metrics namely, Precision, Recall, F Score and Pyramid Precision to compare and evaluate our techniques.

- Precision: It is a measure of how accurate the predictions are. Intuitively, it is the fraction of sentences in the generated summary that are also present in the oracle set and is formally defined as:

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

  where TP stands for True Positive and FP is False Positive

- Pyramid Precision (p): This evaluation scheme is used when multiple reference summaries are available and we used the one defined in [8]. Let $RS(S_i)$ be the number of reference summaries in which the sentence S_i has been included, $GSum$ be the generated summary of length n and Sum_n be any summary of length n. Then, it is defined as:

$$Pyramid\ Precision = \frac{\Sigma_{S_i \in GSum} RS(S_i)}{max\ \Sigma_{S_j \in Sum\_n} RS(S_j)} \tag{5}$$

  Denominator in the above equation is the score of the best possible summary of length n.

- Recall: It is a measure of the ability of algorithm to select the sentences of the summary. Intuitively, it is the fraction of summary sentences in the oracle that we also identify in our generated summary. It is formally defined as:

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

  where FN is False Negative

- F Score: is the harmonic mean of precision and recall. It is defined as:

$$F\ score = 2 \times \frac{precision \times recall}{precision + recall} \tag{7}$$

Rest of the section is organized as follows. In section 4.2 we apply the unsupervised techniques to both the subjects, and compare the unsupervised approaches with the supervised approaches for the *SDS* subject. In section 4.3 we evaluate the impact of applying the proposed noise reduction technique on unsupervised summaries for both *SDS* and *DB2-Bind*. Finally, in section 4.4, we present the results of the goodness of classification of sentences for both *SDS* and *DB2-Bind*.

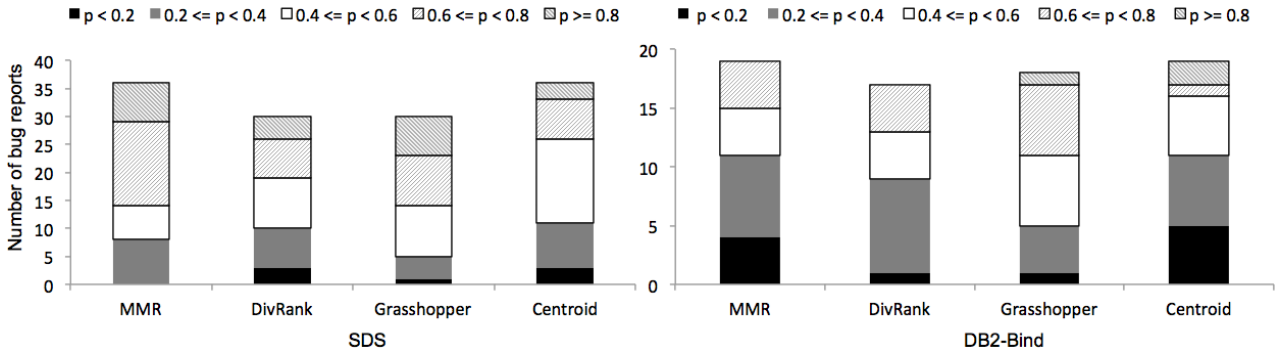

Figure 5: Distribution of bug reports in five segments of average pyramid precision with respect to unsupervised techniques for subjects SDS and DB2-Bind.

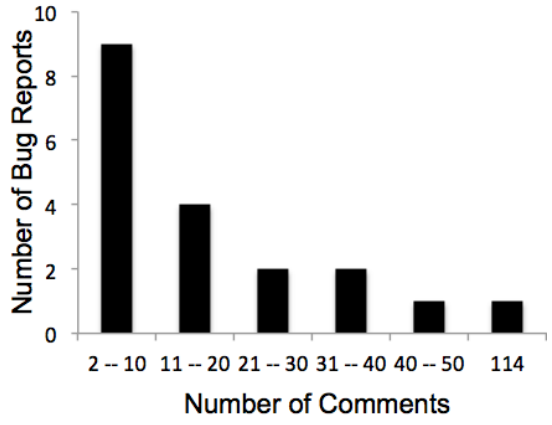

Figure 4: Distribution of bug reports for DB2-Bind with respect to number of comments.

4.2 RQ1 - Feasibility of Unsupervised Techniques for Summarization

In this section, we first evaluate the feasibility of the unsupervised techniques for summarization without applying any noise filtering. Further we compare the results with Supervised techniques for *SDS* subject.

4.2.1 Efficacy of Unsupervised Summarization

All unsupervised algorithms, return a ranked list of sentences from the bug reports. Since our granularity was at sentence level, we generated the summaries for each bug, by picking the same percentage of sentences that the bug had in the reference summaries (oracle set) created manually. Figure 5 shows the results of applying the four unsupervised summarization techniques on the two datasets. X-axis represents the four unsupervised techniques. Each vertical bar represents the number of bug reports covered by that technique and each segment represent the range of pyramid precision.

For *SDS*, *MMR* worked the best by summarizing 22 out of 36 bugs with pyramid precision > 0.6. *Centroid* technique performed the worst by summarizing almost 26 of the total 36 bugs with precision < 0.6. Similarly for *DB2-Bind*, *Grasshopper* worked the best – 7 out of 19 bugs with precision > 0.6 and *Centroid* the worst.

For *SDS*, *DivRank* and *Grasshopper* did not generate summarization for 6 of the bug reports. Similarly for *DB2-Bind*, *DivRank* missed two bug reports and *Grasshopper* missed one. As discussed in the method earlier (Section 3), these two techniques work by applying random walks on a graph. Depending on the nodes in the graph (size of bug report) and the edges (similarity between sentences), the algorithm might not converge in a finite number of iterations or time period. While experimenting we found these were not converging at all for the 9 bug reports overall.

Based on this experiment, we conclude that though *Grasshopper* and *DivRank* are more sophisticated algorithms and at individual bug level might give better efficacy, *MMR* is a safer algorithm to use for bug reports because it guarantees summarization for any bug size in finite time period.

4.2.2 Comparison of Unsupervised Summarization with Supervised Approach

We now compare the results of our unsupervised techniques with three supervised algorithms used by Rastkar et. al[34] on the *SDS* dataset[5]

- EC : This classifier was basically developed for summarizing emails [25] and was trained on Enron email corpus [17]. This was chosen for its similarity to the bug report corpus.
- EMC : This is similar to EC, but was trained on a combination of email threads and meetings [25], where the meeting data comes from AMI meeting corpus [9].
- BRC : This is the only classifier that was trained on bug reports from the same projects as the test bug reports. They used a linear classifier from the Liblinear toolkit[6]

Table 3 gives the numbers for Pyramid Precision, Precision, Recall and F Score for the three supervised techniques and four unsupervised techniques. Based on this data, following observations can be made:

- Even though BRC achieves the highest Pyramid Precision and Precision, both the unsupervised methods *MMR* and *Centroid* out-performed it on the overall F score numbers. *MMR* was almost at par with BRC in terms of pyramid precision.
- The unsupervised method *MMR* performed much better than the EC and the EMC techniques with improvement in all measured metrics. *Centorid* was almost at par in precision and provides better recall and F-Score. This suggests that, probably the email and meeting corpora are not as similar to the bug reports, as was perceived before in [34]. Also, this shows if the domain of the classifier is not similar to the domain of the data, then the returns are lower than using an unsupervised summarizer.
- The unsupervised methods *Grasshopper, DivRank*, when they worked, provided much better recall and F-Score than *BRC* and almost similar levels for precision.

[5]We did not apply the learning approach to *DB2-Bind* dataset, as the implementation of the supervised approach was not available.

[6]http://www.csie.ntu.edu.tw/~cjlin/liblinear/

Table 3: Comparison of the unsupervised approaches against the supervised technique on *SDS* dataset

Technique	Algorithm	Pyramid	Precision	Recall	F Score
Supervised	BRC	.63	.57	.35	.40
	EC	.54	.43	.3	.32
	EMC	.53	.47	.23	.29
Unsupervised	Centroid	.52	.42	.43	.43
	MMR	.60	.47	.49	.48
	Grasshopper	.60	.49	.51	.50
	DivRank	.50	.45	.46	.46

Based on the above, we conclude that unsupervised approaches are viable options to apply to summarize bug reports. They outperform learning based approaches that are not trained on bug reports from similar domain.

4.3 RQ2 - Impact of Noise Identifier on effectiveness of Unsupervised techniques

Figure 6 shows the observed change in average pyramid precision when applying noise reduction. The black bar shows the precision without applying noise reduction. The gray bar show precision when only the *Others* sentences were filtered. White bar shows the summarization precision when we also filtered out the *Code* sentences. The x-axis lists the different summarization algorithms applied. As is evident, noise reduction was effective in both the techniques. However, in the *SDS* dataset, filtering away *Code* and *Others* sentences was more effective. While, on the *DB2-Bind* subject, just filtering the *Others* sentences worked better in three of the four techniques. Among the different summarization techniques, noise reduction impacted *Centroid* the least. This is because, *Centroid* method inherently uses *tf-idf(t,d)* scores of terms to choose the summary sentences and was already filtering out those with low tf-idf value. The maximum pyramid precision value was obtained for *SDS* using *MMR* (0.63) while for *DB2-Bind* using *Grasshopper* (0.61). On average across the four unsupervised techniques applying noise reduction improved the summarization efficacy by 11% for *SDS* and 15% for *DB2-Bind*.

A question that arises is: why did precision decrease in some case when we applied noise filtering. There are two reasons for this: (1) our classification of sentences is not full-proof. It is likely that we classify summary sentences as *Others* and remove them by filtering. The goodness of classification is discussed in next section. (2) *Code* fragments, though they never show up in the summary, they affect the calculation of centrality measure of other sentences. By removing code sentences, some oracle summary sentences, which earlier had a high centrality measure, has low centrality score and hence were removed from the calculated summaries.

One important thing to note is that after applying noise reduction, both *DivRank* and *Grasshopper* generated summaries for all the nine bug reports, which they earlier missed across the two subjects.

4.4 RQ3 - Goodness of Noise Identifier

In this section we evaluate the goodness of noise identifier in terms of how well it is able to auto-classify sentences. We first present our approach to choose the cutoff value for identifying whether a word is a keyword or not. Then we present the correctness of the classification technique.

4.4.1 Cut-off Value for Identifying Keywords

As mentioned in the proposed approach (Section 2.1.4), in order to apply noise reduction, we first need to populate the keyword dictionary. The keyword algorithm requires us to select a *tf-idf(t,d)* cut-off to be used. To calculate the cut-off we followed the following process.

We extracted the list of keywords for both the subjects. The average and median for *tf-idf* scores for the subject *SDS* was 0.85 and 0.86 and for *DB2-Bind* was 0.77 and 0.79 respectively. For these four iterations of cut-off scores 0, 0.7, 0.8 and 0.9, we executed the following process. For each bug, rank each sentence based on the sum of *tf-idf* scores of the keywords present in each sentences. Then pick top 30% of sentences as the summary and compare with the manual summaries provided by the annotators. We refer to this process as the *naive summarization*.

The average precision, recall and F Score of the *naive summarization* across all annotators for both subjects is presented in Table 4. For *SDS*, there was no change in any of the metrics across the various cut-offs indicating that sentences in the summary oracle had very high *tf-idf* scores. Similarly, for *DB2-Bind*, there was hardly any difference in precision / recall / F score between 0 and 0.8 cut-off. With 0.9 cut-off there was larger dip of .04 in recall and .02 in F score, when compared to 0.8 cut-off. Based on this analysis, we chose to apply 0.8 as *tf-idf(t,d)* cut-off score for both the subjects and populated the keyword dictionary.

4.4.2 Goodness of Classification

Table 5 gives the details on the classification of the sentences across the four classes *Question*, *Code*, *Investigative* and *Others* for both the subjects. Across all bug reports, almost 22% of sentences for *SDS* and 54% for *DB2-Bind* are marked as *Others*. Also, 5.6% of sentences for *SDS* and 14.5% for *DB2-Bind* are marked as *Code*. In order to confirm the goodness of our noise reduction approach, we calculated the overlap between the identified *Others* sentences and the summary sentences in the oracle.

Figure 7 presents the distribution of bug reports across percentage of sentences classified as *Others*, which were actually used by the annotators in their summary. The X-axis represent the percentage distribution and the Y-axis the number of bug reports. Each segment in the vertical represents the number of bug reports per subject. The total number of bug reports across both the subjects for each percentage overlap is listed on top of each bar.

As it is evident from the plot, for *DB2-Bind*, mo15 of the total 19 bug reports (78%) have only 10% or less summary sentences classified as *Others*. Similarly for 77% of *SDS* bug reports (28 out of 36) only 20% or less summary sentences are classified as *Others*. For 7 of the bug reports across subjects no summary sentences were classified as *Others*. However for just 1 bug report of *SDS*, the percentage overlap was more than 40%.

To summarize, the keyword based classification scheme, was not full-proof. For *DB2-Bind*, 30 out of the 371 sentences (i.e. 8%) in the oracle summary, were incorrectly classified as *Others*. Similarly, F=for *1*95 out of 840 oracle summary sentences (23%) were incorrectly classified. However this is acceptable as (1) even with less than 100% correctness we did see an improvement in precision in both *SDS* and *DB2-Bind* after noise filtering and (2) typically most automatic classification techniques are not completely fool proof.

5. DISCUSSION

In this section we discuss our choice of unsupervised summarization algorithms and the threats to validity of our experiments.

5.1 Choice of Unsupervised Summarization Algorithms

The summarization algorithms that we used, namely, *Centroid*, *MMR*, *Grasshopper* and *DivRank* form a representative set of unsu-

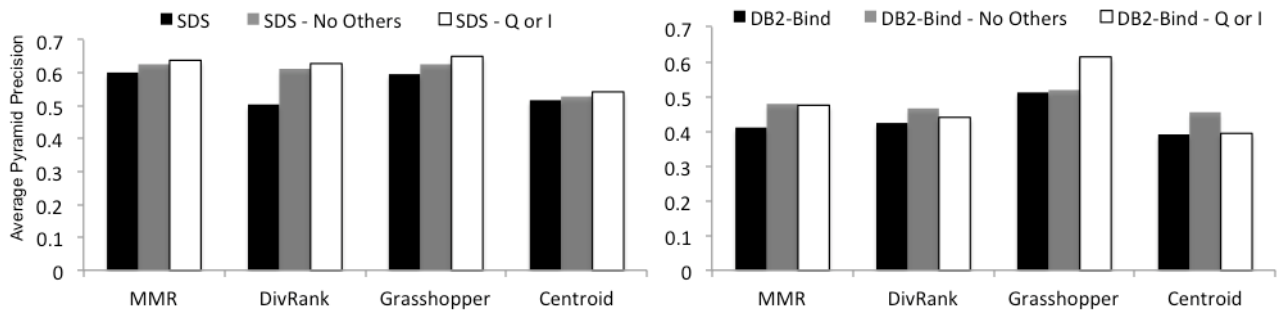

Figure 6: Effect of Noise reduction in Unsupervised Summarization. Average Pyramid Precision of four unsupervised techniques using noise reduction for *SDS* (left) and *DB2-Bind* (right).

Table 4: Average Precision, Recall and F Score for different *tfidf* cut-offs applied for Naive Summarization

Subject	Cut-off	Precision	Recall	F Score
SDS	0	0.34	0.28	0.30
	0.7	0.34	0.28	0.30
	0.8	0.34	0.28	0.30
	0.9	0.34	0.28	0.30
DB2-Bind	0	0.23	0.48	0.31
	0.7	0.22	0.47	0.29
	0.8	0.21	0.45	0.28
	0.9	0.2	0.41	0.26

Table 5: Classification of sentences

Subjects	Total	*Question*	*Code*	*Investigative*	*Others*
SDS	2361	94	134	1614	519
DB2-Bind	6304	215	917	1720	3452

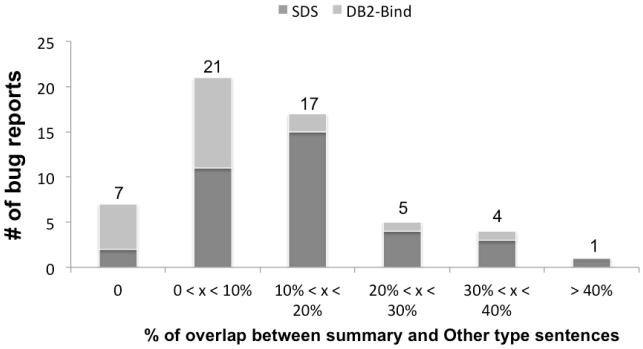

Figure 7: Distribution of bug reports with respect to percentage of sentences classified as *Others*, which were actually used by Annotators in their summary.

pervised summarization algorithms in the existing literature, starting from simple earlier methods to recent sophisticated methods. Out of these four methods, *Centroid* is the least sophisticated and is expected to perform the worst, which is evident from the experiments. However, it is still interesting to note that this simple method is on par with the supervised EC and EMC methods (refer Table 3).

Also, according to the literature and as discussed in Section 2.2, *DivRank* is expected to perform better than *Grasshopper*, which is in turn expected to be better than *MMR*. However, our experiments show that *MMR* performed the best, when applied without noise reduction. This could be because, *DivRank's* performance was tuned to the social and citation (author and paper) network data (in [23]), and may not be reproducible on all datasets. *DivRank* and *Grasshopper* also missed generating summaries for bugs where the convergence was not attained owing to the graph size. Increasing the iteration cut-off in the algorithm or reducing the size of the bug reports can help the algorithm complete. The latter (which required a research solution) is precisely what we achieved through noise reduction. Once the noise reduction was applied, the size of the bug reports largely reduced and for both these unsupervised techniques, we were able to generate summaries for all the bugs in the datasets.

5.2 Threats to Validity

Threats to external validity arise when the observed results cannot be generalized to other experimental setups. In our experiments we tried to limit this threat by evaluating a bug corpus of 55 bug reports that had equal number of bug reports from 4 different open-source eclipse projects and one internal industry project. The number of sentences required to be summarized per bug report also varied from as low as 17 to a maximum of 3766. However, we cannot conclude how our observations might generalize to other projects. More empirical evaluation with other projects is required to evaluate how our results may generalize. These we intend to do in future research.

Threats to internal validity arise when factors affect the dependent variables (the data described in Section 4) without the researchers' knowledge. In our study, such factors are errors in implementation of noise reduction. Moreover, we used certain heuristics in classification of sentences into classes. These include:

- Patterns for identifying code and question sentences
- Cut-off of three keywords chosen to classify a sentence into investigative sentence
- Cut-off of 0.8 chosen on *tf-idf* score, to identify keywords.

These heuristics were based on manual inspection of bug reports in the test subjects and might not be generalizable to an unseen population of bugs. They need re-configuration depending on the subject to which noise reduction is applied.

6. RELATED WORK

Summarization of documents is a well researched area, starting from the first work published in 1958 by Luhn [20] for creating literature abstracts. This was followed by a slew of papers that improved the summarization method from the simple tf-idf based techniques to more complex natural language processing and machine learning based methods.

Summarization has been applied successfully to many domains like news articles [31], social media [19], medical documents [3], videos [16], and audio [40], in addition to technical documents. It has also been incorporated into various products like IBM's Intelligent Miner for Text[1] and Microsoft's Office Suite[2]

Summarization techniques have been broadly classified into two types - Extractive and Abstractive, as discussed in Section 2.2. One

of the earliest methods for abstractive summarization is the SUMMONS system [29] which extends a template driven document understanding method. Abstractive methods have also used language generation models in the context of summarizing arguments [7] and controversial statements [6].

Another broad classification of summarization techniques is as supervised and unsupervised methods. Supervised methods like those proposed in [26, 43, 45, 18] train a decision tree, SVM etc to learn sentences that belong to the summary. However, obtaining training data is usually costly, which has led to unsupervised methods being more popular.

Summarization methods proposed initially, like [24, 13] considered only centrality of the chosen sentences. However, with the introduction of the concept of diversity in [5], there has been considerable work in this area like [46, 47]. More recent works have proposed using different types of random walks on a graph constructed out of the document for summarization like [48, 23, 12].

Summarizing bug reports have been previously studied only in [34] where they used supervised methods. However, text analytics and machine learning methods have been previously applied to software repositories and bug reports like [4] for assigning reports to the correct person, [42] for estimating the time taken to solve a bug, [35, 41] for detecting duplicate reports, etc. Summarizing bug reports using unsupervised techniques and dealing with noise in the reports have not been studied before.

A different but related problem is extracting problem resolution information from various datasources like online discussion forums [10, 33], emails [36], problem tickets [44] etc. This is similar in the sense that, a summary can be constructed out of the extracted solutions. However, these tasks are more tuned to extracting solutions and are not constrained by any summary size; nor do they give emphasis to diversity.

A related research problem is the issue of summarizing source code. Sridhara et al. developed novel heuristics to automatically summarize a Java method in the form of natural language comments [38, 37]. They also developed heuristics to automatically detect and describe high level actions within a method [39].

7. CONCLUSION

In this paper, we evaluated four unsupervised techniques for bug report summarization. We compared the efficacy of the techniques with supervised approaches. *MMR*, *DivRank* and *Grasshopper* algorithms, worked at par with the best of the supervised approach. For both the subjects, the efficacy of the unsupervised techniques improved by applying noise identifier and filtering out sentences classified as *Useless* and *Code*. Importantly, two of the algorithms *DivRank* and *Grasshopper* which did not converge for 9 bug reports, converged successfully for them and generated summaries once noise identifier based filtering was applied.

One area of future work is to see if we can improve the precision of summarization approaches so as to be able to auto-extract *Frequently Asked Questions* from a bug repository. Another direction for future work is to evaluate if the text summarization approaches mentioned in this paper can be used for code summarization. The heuristics discussed in [38, 37] can be used to summarize methods to natural language. Given this natural language text, augmented with comment text, we can consider generating class and package level summaries.

Acknowledgment

We would like to thank Ananthkumar Peddi and Randeep Ghosh of IBM DB2 team for manually summarizing the DB2 bug reports. Thanks are also due to Sarah Rastkar and Gail C. Murphy for making available the bug report corpus and the annotated summaries. Finally, we would also like to thank Giriprasad Sridhara, Rema Ananthanarayanan and Karthik Visweswariah of IBM Research - India, for their valuable comments and suggestions.

References

[1] Intelligent miner for text. `http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?subtype=ca\&infotype=an\&appname=iSource\&supplier=897\&letternum=ENUS298-447`, 2012.

[2] Microsoft office suite. `http://office.microsoft.com/en-us/word-help/automatically-summarize-a-document-HA010255206.aspx`, 2012.

[3] Stergos Afantenos, Vangelis Karkaletsis, and Panagiotis Stamatopoulos. Summarization from medical documents: a survey. *Artificial Intelligence in Medicine*, 33:157–177, 2005.

[4] John Anvik, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 361–370, 2006.

[5] Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proc. 21st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '98, pages 335–336, 1998.

[6] Giuseppe Carenini and Jackie Chi Kit Cheung. Extractive vs. nlg-based abstractive summarization of evaluative text: the effect of corpus controversiality. In *Proceedings of the Fifth International Natural Language Generation Conference*, INLG '08, pages 33–41, 2008.

[7] Giuseppe Carenini and Johanna D. Moore. Generating and evaluating evaluative arguments. *Artificial Intelligence*, 170:925–952, 2006.

[8] Giuseppe Carenini, Raymond T. Ng, and Xiaodong Zhou. Summarizing emails with conversational cohesion and subjectivity. In *ACL '08*, pages 353–361, 2008.

[9] Jean Carletta, Simone Ashby, Sebastien Bourban, Mike Flynn, Thomas Hain, Jaroslav Kadlec, Vasilis Karaiskos, Wessel Kraaij, Melissa Kronenthal, Guillaume Lathoud, Mike Lincoln, Agnes Lisowska, and Mccowan Wilfried Post Dennis Reidsma. The ami meeting corpus: A pre-announcement. In *Proceedings of Machine Learning in Medical Imaging*, pages 28–39, 2005.

[10] Gao Cong, Long Wang, Chin-Yew Lin, Young-In Song, and Yongheng Sun. Finding Question-Answer Pairs from Online Forums. In *SIGIR*, 2008.

[11] D. Cubranic and G.C. Murphy. Hipikat: Recommending pertinent software development artifacts. 2003.

[12] Avinava Dubey, Soumen Chakrabarti, and Chiranjib Bhattacharyya. Diversity in ranking via resistive graph centers. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 78–86, 2011.

[13] Günes Erkan and Dragomir R. Radev. Lexrank: graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22:457–479, 2004.

[14] J. Fleiss et al. Measuring nominal scale agreement among many raters. volume 76, pages 378–382, 1971.

[15] Udo Hahn and Inderjeet Mani. The challenges of automatic summarization. *Computer*, 33:29–36, 2000.

[16] Bohyung Han, Jihun Hamm, and J. Sim. Personalized video summarization with human in the loop. In *IEEE Workshop on Applications of Computer Vision*, WACV 2011, pages 51 –57, 2011.

[17] Bryan Klimt and Yiming Yang. Introducing the enron corpus. In *CEAS '04*, 2004.

[18] Liangda Li, Ke Zhou, Gui-Rong Xue, Hongyuan Zha, and Yong Yu. Enhancing diversity, coverage and balance for summarization through structure learning. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 71–80, 2009.

[19] Yu-Ru Lin, H. Sundaram, and A. Kelliher. Summarization of large scale social network activity. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, ICASSP 2009, pages 3481 –3484, 2009.

[20] H. P. Luhn. The automatic creation of literature abstracts. *IBM Journal of Research Development*, 2:159–165, 1958.

[21] Debapriyo Majumdar, Rose Catherine, Shajith Ikbal, and Karthik Visweswariah. Privacy protected knowledge management in services with emphasis on quality data. In *20th ACM Conference on Information and Knowledge Management (CIKM)*, 2011.

[22] C. D. Manning, P. Raghavan, and H. Sch§tze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. URL `http://www-csli.stanford.edu/~hinrich/information-retrieval-book.html`.

[23] Qiaozhu Mei, Jian Guo, and Dragomir Radev. Divrank: the interplay of prestige and diversity in information networks. In *Proc. 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 1009–1018, 2010.

[24] Rada Mihalcea and Paul Tarau. TextRank: Bringing Order into Texts. In *Conference on Empirical Methods in Natural Language Processing*, 2004.

[25] Gabriel Murray and Giuseppe Carenini. Summarizing spoken and written conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 773–782, 2008.

[26] Tadashi Nomoto and Yuji Matsumoto. Supervised ranking in open-domain text summarization. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 465–472, 2002.

[27] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.

[28] R. Pemantle. Vertex reinforced random walk. *Probability Theory and Related Fields*, pages 117–136, 1992.

[29] Dragomir R. Radev and Kathleen R. McKeown. Generating natural language summaries from multiple online sources. *Computational Linguistics*, 24:470–500, 1998.

[30] Dragomir R. Radev, Hongyan Jing, and Malgorzata Budzikowska. Centroid-based summarization of multiple documents: sentence extraction utility-based evaluation, and user studies. *CoRR*, cs.CL/0005020, 2000.

[31] Dragomir R. Radev, Sasha Blair-goldensohn, Zhu Zhang, and Revathi Sundara Raghavan. NewsInEssence: A system for domain-independent, real-time news clustering and multi-document summarization. In *In Proceedings of the Human Language Technology Conference (HLT-01*, 2001.

[32] Dragomir R. Radev, Hongyan Jing, Malgorzata Stys, and Daniel Tam. Centroid-based summarization of multiple documents. *Information Processing and Management*, 40:919–938, November 2004.

[33] Preethi Raghavan, Rose Catherine, Shajith Ikbal, Nanda Kambhatla, and Debapriyo Majumdar. Extracting problem and resolution information from online discussion forums. In *Proc. of International Conference on Management of Data*, COMAD '10, 2010.

[34] Sarah Rastkar, Gail C. Murphy, and Gabriel Murray. Summarizing software artifacts: a case study of bug reports. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 505–514, 2010.

[35] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. Detection of duplicate defect reports using natural language processing. In *ICSE '07*, pages 499–510, 2007.

[36] Lokesh Shrestha and Kathleen McKeown. Detection of Question-Answer Pairs in Email Conversations. In *Proc. International Conference On Computational Linguistics*, 2004.

[37] Giriprasad Sridhara. *Automatic Generation of Descriptive Summary Comments for Methods in Object-Oriented Programs*. PhD thesis, University of Delaware, 2012.

[38] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K. Vijay-Shanker. Towards automatically generating summary comments for java methods. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ASE '10, pages 43–52, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0116-9. doi: http://doi.acm.org/10.1145/1858996.1859006. URL `http://doi.acm.org/10.1145/1858996.1859006`.

[39] Giriprasad Sridhara, Lori Pollock, and K. Vijay-Shanker. Automatically detecting and describing high level actions within methods. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 101–110, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0445-0. doi: 10.1145/1985793.1985808. URL `http://doi.acm.org/10.1145/1985793.1985808`.

[40] A. Waibel, M. Bett, F. Metze, K. Ries, T. Schaaf, T. Schultz, H. Soltau, Hua Yu, and K. Zechner. Advances in automatic meeting record creation and access. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP '01, pages 597–600, 2001.

[41] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *ICSE '08*, pages 461–470, 2008.

[42] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *Fourth International Workshop on Mining Software Repositories*, ICSE Workshops MSR '07, 2007.

[43] Kam-Fai Wong, Mingli Wu, and Wenjie Li. Extractive summarization using supervised and semi-supervised learning. In *Proceedings of the 22nd International Conference on Computational Linguistics*, COLING '08, pages 985–992, 2008.

[44] Ruchi Mahindru Xing Wei, Anca Sailer and Gautam Kar. Automatic Structuring of IT Problem Ticket Data for Enhanced Problem Resolution. In *IFIP/IEEE International Symposium on Integrated Network Management*, 2007.

[45] Yisong Yue and Thorsten Joachims. Predicting diverse subsets using structural svms. In *ICML'08*, pages 1224–1231, 2008.

[46] Cheng Xiang Zhai, William W. Cohen, and John Lafferty. Beyond independent relevance: methods and evaluation metrics for subtopic retrieval. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, SIGIR '03, pages 10–17, 2003.

[47] Benyu Zhang, Hua Li, Yi Liu, Lei Ji, Wensi Xi, Weiguo Fan, Zheng Chen, and Wei-Ying Ma. Improving web search results using affinity graph. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 504–511, 2005.

[48] Xiaojin Zhu, Andrew B. Goldberg, Jurgen Van, and Gael David Andrzejewski. Improving diversity in ranking using absorbing random walks. In *Physics Laboratory âĂŞ University of Washington*, pages 97–104, 2007.