Duplicate Bug Report Detection with a Combination of Information Retrieval and Topic Modeling

Anh Tuan Nguyen,
Tung Thanh Nguyen,
Tien N. Nguyen
Iowa State University, USA
{anhnt,tung,tien}@iastate.edu

David Lo
Singapore Management University, Singapore
davidlo@smu.edu.sg

Chengnian Sun
National University of Singapore, Singapore
suncn@comp.nus.edu.sg

ABSTRACT

Detecting duplicate bug reports helps reduce triaging efforts and save time for developers in fixing the same issues. Among several automated detection approaches, text-based information retrieval (IR) approaches have been shown to outperform others in term of both accuracy and time efficiency. However, those IR-based approaches do not detect well the duplicate reports on the same technical issues written in different descriptive terms.

This paper introduces DBTM, a duplicate bug report detection approach that takes advantage of both IR-based features and topic-based features. DBTM models a bug report as a textual document describing certain technical issue(s), and models duplicate bug reports as the ones about the same technical issue(s). Trained with historical data including identified duplicate reports, it is able to learn the sets of different terms describing the same technical issues and to detect other not-yet-identified duplicate ones. Our empirical evaluation on real-world systems shows that DBTM improves the state-of-the-art approaches by up to 20% in accuracy.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

General Terms

Algorithms, Documentation, Management, Reliability

Keywords

Duplicate Bug Reports, Topic Model, Information Retrieval

1. INTRODUCTION

Bug fixing is vital in producing high-quality software products. Bug fixing happens in both development and post-release time. In either case, the developers, testers, or end-users run a system and find its incorrect behaviors that do not conform to their expectation and the system's requirements. Then, they report such occurrences in a bug report, which are recorded in an issue-tracking database.

Generally, there are many users interacting with a system and reporting its issues. Thus, a bug is occasionally reported by more than one reporters, resulting in *duplicate bug reports*. Detecting whether a new bug report is a duplicate one is crucial. It helps reduce the maintenance efforts from developers (e.g. if the bug is already fixed). Moreover, duplicate reports provide more information in the bug fixing process for that bug (e.g. if the bug is not yet fixed) [4].

To automate the detection of duplicate bug reports, several approaches have been introduced. Early approaches have applied information retrieval (IR) to this problem with Vector Space Model (VSM) in which a bug report is modeled as a vector of textual features computed via Term Frequency-Inverse Document Frequency (Tf-Idf) term weighting measurement [14, 23]. To improve the detection accuracy, natural language processing (NLP) has been combined with those IR methods [23]. Execution trace information on the reported bugs in the bug reports is also used in combination with NLP [26]. However, execution traces might not be available in all bug reports. Another predominant approach to this problem is machine learning (ML). Jalbert and Weimer [16] use a binary classifier model and apply a linear regression over textual features of bug reports computed from their terms' frequencies. Support Vector Machine (SVM) was utilized by Sun *et al.* [25]. To train an SVM classifier, all pairs of duplicate bug reports are formed and considered as the positive samples and all other pairs of non-duplicate bug reports are used as the negative ones. The key limitation of ML approaches is their low efficiency.

The recent work by Sun *et al.* has shown that REP [24], an advanced IR approach, outperformed state-of-the-art ML approaches in term of both accuracy and time efficiency. It is customized from BM25F [22] to take into account the long bug reports and the meta-data such as the reported product, component, and version. The key assumption in REP is based on high textual similarity between duplicate bug reports. However, in practice, it is popular that the bug reports can be filed by multiple reporters who could describe about the same technical issue(s) in different phenomena via different terms. With different input data, usage environments or scenarios, an erroneous behavior might be exposed as different phenomena (e.g. different outputs, traces, or screen views). Moreover, different reporters might use different terminologies and styles, or write about different phenomena to describe the same issue(s). Thus, duplicate bug

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ASE'12, September 3–7, 2012, Essen, Germany
Copyright 2012 ACM 978-1-4503-1204-2/12/09 ...$15.00

Authorized licensed use limited to: Nanjing University. Downloaded on August 07,2020 at 06:06:37 UTC from IEEE Xplore. Restrictions apply.

ID:000002; **CreationDate**:Wed Oct 10 20:34:00 CDT 2001; **Reporter**:Andre Weinand
Summary: Opening repository resources doesn't honor type.
Description:Opening repository resource always open the default text editor and doesn't honor any mapping between resource types and editors. As a result it is not possible to view the contents of an image (*.gif file) in a sensible way.

Figure 1: Bug Report BR2 in Eclipse Project

reports might not be very textually similar. In those cases, REP does not detect them well.

This paper introduces DBTM, a duplicate bug report detection model that takes advantage of not only IR-based features but also topic-based features from our novel topic model, which is designed to address textual dissimilarity between duplicate reports. In DBTM, a bug report is considered as a textual document describing one or more technical issues/topics in a system. Duplicate bug reports describe the same technical issue(s) even though the issue(s) is reported in different terms. In our topic model, we extend Latent Dirichlet Allocation (LDA) [6] to represent the topic structure for a bug report as well as the duplication relations among them. Two duplicate bug reports must describe about the shared technical issue(s)/topic(s) in addition to their own topics on different phenomena. The topic selection of a bug report is affected not only by the topic distribution of that report, but also by the buggy topic(s) for which the report is intended. We also apply Ensemble Averaging technique [11] to combine IR and topic modeling in DBTM. We use Gibbs sampling [13] to train DBTM on historical data with identified duplicate bug reports and then detect other not-yet-identified duplicate ones.

Our empirical evaluation results on large, real-world systems show that DBTM with both topic-based and textual features can improve the state-of-the-art approach REP [24] by up to 20% in accuracy. For a new-coming bug report B, if it is a duplicate, in 57% of the time, DBTM can correctly identify its duplicate bug report with only a single recommendation. In 76% of the time, a developer just needs to examine the resulting list of 5 recommended reports and (s)he will be able to identify the duplicate one. The number is 82% if DBTM recommends 10 bug reports for B. Importantly, DBTM is very efficient in both training and predicting time. The contributions of this paper include:

1. DBTM, a combined model taking the strength of both topic-based features from a novel topic model and textual features from an IR model, BM25F. Our new topic model captures semantically the technical issues in the bug reports and formulates the semantic similarity measure among duplicate reports based on such topic structures;
2. Algorithms for training/detecting duplicate reports;
3. An evaluation on DBTM's accuracy and efficiency.

Next section presents a motivating example. Section 3 describes the details of DBTM. Section 4 presents our training and detection algorithms. Section 5 discusses our evaluation. Related work is in Section 6. Conclusion appears last.

2. MOTIVATING EXAMPLE

Let us begin with an example of duplicate bug reports that motivates our approach. Generally, a bug report is a record in a bug-tracking database, containing several descriptive fields about the reported bug(s). Important fields in a bug report include a unique identification number of the report (ID), creation time (CreationDate), the reporting person (Reporter), and most importantly, a short summary (Summary) and a full description (Description) of the bug(s).

ID:009779; **CreationDate**:Wed Feb 13 15:14:00 CST 2002; **Reporter**:Jeff Brown
Resolution:DUPLICATE
Summary: Opening a remote revision of a file should not always use the default text editor.
Description: OpenRemoteFileAction hardwires the editor that is used to open remote file to org.eclipse.ui.DefaultTextEditor instead of trying to find an appropriate one given the file's type.
You get the default text editor regardless of whether there are registered editors for files of that type – even if it's binary. I think it would make browsing the repository or resource history somewhat nicer if the same mechanism was used here as when files are opened from the navigator. We can ask the Workbench's IEditorRegistry for the default editor given the file name. Use text only as a last resort (or perhaps because of a user preference).

Figure 2: Bug Report BR9779, a Duplicate of BR2

Observations on a bug report Figure 1 displays an example of an already-fixed bug report in Eclipse project. This bug report was assigned with ID 2 and reported on 10/10/2001 by Andre Weinand for a bug on Eclipse v2.0. It described that the system always use the default text editor to open and display any resource file (e.g. a GIF image) stored in the repository despite its type. Analyzing the contents of BR2, we have the following observations:

1. This bug report is about two technical functions in Eclipse: artifact manipulation (MAN) and resource versioning (VCM). Consulting Eclipse's documentation, we found that MAN involves operations such as *opening*, *viewing*, *editing*, and *saving* files/resources. VCM involves operations such as *connecting*, *committing*, *updating* to repositories, etc.
2. The bug occurred in the code implementing MAN. That is, the operation *opening* a resource file in the repository was incorrectly implemented. We can consider MAN as a technical issue reported in BR2. We can see that Andre Weinand reported that issue in the context of opening version repository resource (VCM). He also described the phenomenon in the context of opening an GIF image file.
3. In BR2, the technical function MAN can be recognized in its contents via the words that are relevant to MAN such as editor, view, content, resource, file, and text. Similarly, the relevant terms to VCM in the report are repository, resource, and file. Considering bug reports as textual documents, we can view the described technical issues as their **topics**.

Observations on a duplicate bug report Figure 2 shows bug report #9779, filed on 02/13/02 by a different person, Jeff Brown. This report was determined by Eclipse's developers as reporting the same bug as in BR2. Analyzing the content of BR9779 and comparing it to BR2, we can see that

1. BR9779 also reported on the same bug in the technical function MAN (i.e. file manipulation). Jeff Brown reported the issue in the context of opening remote files with more detailed information on the code realizing that function: OpenRemoteFileAction, the class responsible for opening a remote file, is directly associated with org.eclipse.ui.DefaultTextEditor, i.e. it always uses the default editor to open a remote file. He provided a suggestion for fixing by using Workbench's

Authorized licensed use limited to: Nanjing University. Downloaded on August 07,2020 at 06:06:37 UTC from IEEE Xplore. Restrictions apply.

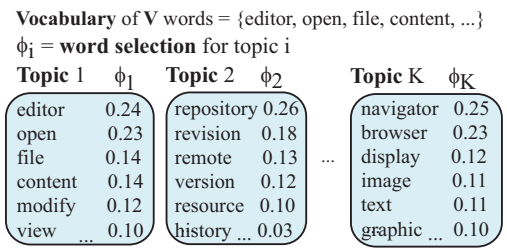

Figure 3: Topic and Word Selection [6]

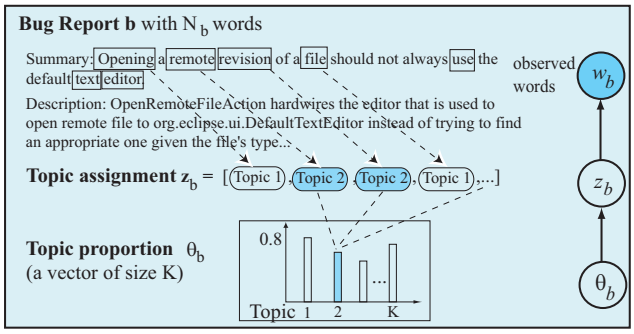

Figure 4: Document Modeling [6]

IEditorRegistry for the default editor given the filename. He also suggested the same mechanism for browsing a resource history in the context of a navigator.

2. In addition to the similar terms used to describe about MAN in both reports (e.g. open, editor, file), there are different terms expressing similar meaning such as honor, mapping, resource type, sensible way in BR2, and appropriate, registered editor, file type in BR9779. The terms for VCM are different in BR9779 and in BR2, e.g. remote, revision, history. Importantly, due to additional information, new terms/topics are used in BR9779 (e.g. registered editor, user preference, navigator).

Implications Detecting duplicate bug reports has benefits in software maintenance. Duplicate bug reports filed by people with different points of view and experience could provide different kinds of information about the bug(s), thus, help in the fixing process. Importantly, detecting such duplications will help avoid redundant bug fixing efforts.

Due to the different contexts and phenomena in which the same bug were exposed and discovered by different reporters, the technical issue could be reported with different terms. Duplicate reports describe the same technical issue. However, reporters might discuss other relevant topics and phenomena, and provide the insights on the bug including suggested fixes and relevant technical functions. Those observations suggest that the detection of duplicate bug reports could rely not only on the technical terms, but also on the **technical topics** in the reports. Intuitively, topics are *latent*, *semantic* features, while terms are *visible, textual* features of the reports. They would complement each other, and could be combined to achieve higher detection accuracy.

We develop Duplicate Bug report Topic Model, DBTM, a duplicate bug report detection model that takes advantage of both term-based and topic-based features. In DBTM, a bug report is viewed as a textual document describing one or more technical issues in a system. Duplicate bug reports describe the same technical issues even though the issues might be reported in different terms. We use Ensemble Averaging [11] to combine IR and topic modeling in DBTM.

3. TOPIC MODELING OF DUPLICATE BUG REPORTS

In our approach, a system is considered to have K technical aspects/functionality. Each aspect of the system is considered as a topic, represented via certain words/terms. Among them, some aspects are incorrectly implemented with respect to the system's requirements, thus, causing the bugs/issues being reported. The bug database is considered as a collection of bug reports. Each bug report is considered as a textual document containing a number of words/terms to report on one or multiple technical issues. From now on, we use the terms "aspect" and "topic" interchangeably. The same treatment is for "bug report" and "document".

Our model, DBTM, is the combination of two components: our novel topic model, *T-Model*, and BM25F model. T-Model is an extension of LDA [6]. Let us first summarize the idea of LDA and then present our extension to specialize it to support the detection of duplicate bug reports.

3.1 Topic Modeling with LDA

3.1.1 Vocabulary, Topic, and Word Selection

In LDA, the words in all bug report documents under consideration are collected into a common ***vocabulary*** Voc of size V. To describe about a topic, one might use different words drawn from that vocabulary. Thus, each word in Voc has a different usage frequency in describing a topic k, and a topic can be described via one or multiple words.

To capture that, LDA uses a ***word-selection vector*** ϕ_k of size V for the topic k. Each element of the vector ϕ_k represents the probability of the corresponding word at that element's position in Voc that is used to describe the topic k. Each element v in ϕ_k has a value in [0-1]. For example, for topic 1, $\phi_1 = [0.24, 0.23, 0.14, ...]$ (Figure 3). That is, the probability for the first word in Voc to be used in describing the topic k is 24% while that for the second word is 23%, and so on. A ***topic*** is represented as a set of words with their probabilities (Figure 3). Putting together all vectors ϕ_ks for all K topics, we will have a $K \times V$ matrix ϕ called ***per-topic word distribution*** that represents the word selection for all topics. Note that ϕ is meaningful for the entire collection of all bug reports, rather than for an individual document.

3.1.2 Bug Report

All the texts from the descriptions and summaries in a bug report are extracted to form the *words* of the textual document b, which describe the technical issue(s) (Figure 4). The document b contains N_b words. LDA associates to each document b two key parameters:

a) Topic Assignment Vector z_b. Each of the N_b positions in document b is considered to describe one technical topic. Thus, topic assignment vector z_b for b has the length of N_b. Each element of the vector z_b is an index to one topic.

b) Topic Proportion θ_b. A document b can describe about multiple topics. Thus, LDA associates to each document b a topic proportion θ_b to represent the significance of all topics in b. θ_b for a document b is represented by

Authorized licensed use limited to: Nanjing University. Downloaded on August 07,2020 at 06:06:37 UTC from IEEE Xplore. Restrictions apply.

a vector with K elements, each of which is a value within [0-1] to model the proportion of one topic in document b. Each value refers to one topic and the total of those values is 100%. The higher the value $\theta_b[k]$ is, the more words on topic k exist in the document b. For example, in the bug report BR9779, if $\theta_b = [0.20, 0.24, 0.12, ...]$, 20% of all words in b are about file editing, 24% are about file versioning, etc.

3.1.3 Generative Process

LDA belongs to a type of machine learning called *generative model*. From its generative perspective, a bug report b is viewed as an "instance" generated by a "machine" with 3 aforementioned variables z_b, θ_b, ϕ (Figure 4). Given a document b of size N_b, the machine generates the vector z_b describing the topic of every position in the document b based on the topic proportion θ_b of b. For each position, it then generates a word w_b based on the topic assigned to that position and the per-topic word distribution ϕ_i corresponding to that topic. This is called a generative process.

The words in the documents in a project's history are the observed data. One can train the LDA model with historical data to derive those three parameters to fit the best with the observed data. As a new document b_{new} comes, with the learned parameters, LDA derives the topic assignment $z_{b_{new}}$ and the proportion $\theta_{b_{new}}$ of those topics for b_{new}.

3.2 T-Model for Duplicate Bug Reports

To support the detection of duplicate bug reports, we specifically develop a novel topic model, called *T-Model*. Figure 5 shows the graphical notation of T-Model. Our idea is as follows. Each bug report b_i is modeled by a LDA, which is represented via three parameters: topic proportion θ_{b_i}, topic assignment z_{b_i}, and the selected terms w_{b_i}. While θ_{b_i} and z_{b_i} are latent, the terms w_{b_i} are observable and determined by the topic assignment z_{b_i} and word selection ϕ.

One or more technical functions in the system were incorrectly implemented and reported in multiple duplicate bug reports. The shared technical issue(s) F in those reports is considered as topic(s) and its topic distribution/proportion is denoted by θ_F (Figure 5). Let us use b_1,..., b_M to denote M duplicate bug reports for the shared technical issue F. Those M reports must describe that technical topic. However, in addition to that shared topic, they might also describe other topics. The local topics for each bug report b_i are modeled by the topic proportion θ_{b_i}. Examples of the local topics are image files in BR2 and navigator in BR9779.

The topic assignment z_{b_i} in each bug report b_i is affected by both the topic proportions from itself (θ_{b_i}) and from the buggy topic (θ_F). Thus, in Figure 5, there are dependencies from θ_F to each of the topic assignment z_{b_i}s of the duplicate bug reports b_1 to b_M.

The combined topic proportion $\theta^*_{b_i}$ for a bug report b_i is a combination of its local topic proportion θ_{b_i} and topic proportion θ_F of the shared technical topic. In T-Model, $\theta^*_{b_i} = \theta_{b_i} \times \theta_F$ (Hadamard product). If a topic k has high proportion in both θ_{b_i} and θ_F, it also has a high proportion in $\theta^*_{b_i}$. We use hyper parameters α and β as in LDA. α is the parameter of the uniform Dirichlet prior on topic distributions θ_{b_i} and θ_F. β is the parameter of the uniform Dirichlet prior on the per-topic word selection distribution ϕ.

The parameters of the T-Model can be learned from training stage and then used in predicting stage to estimate the topics of bug reports and to detect the duplicate ones.

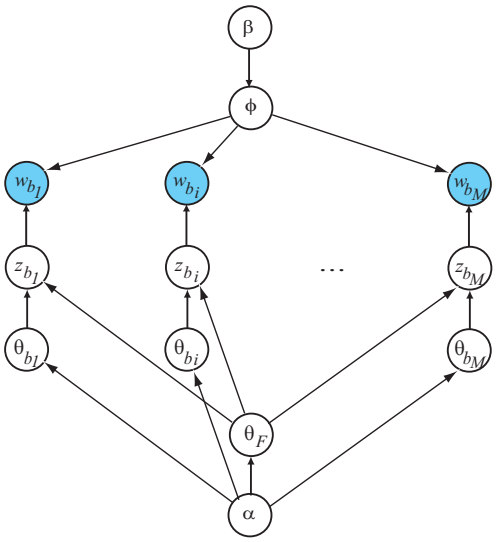

Figure 5: Topic Model for Duplicate Bug Reports

For training, DBTM will be trained from historical data including bug reports and their duplication information. The observed words of bug reports and duplicate relations among them will be used to estimate the topic assignment vectors of all bug reports, the topic proportion of the shared technical issue(s), and the local topic proportions of the bug reports. The variables will be trained to make the model fit most with both the report contents and the duplicate relations. Our training algorithm will be presented in Section 4.

For prediction, we apply DBTM to a new bug report b_{new}. It uses the trained parameters to estimate the topic proportion $\theta_{b_{new}}$ of b_{new}. $\theta_{b_{new}}$ is used to find groups of duplicate bug reports which could share technical issue(s), i.e having high topic proportion similarity, and therefore are potentially duplicate of b_{new}. The topic proportion similarity between b_{new} and a duplicate report group G is measured as:

$$\mathsf{topicsim}(b_{new}, G) = \max_{b_i \in G}(\mathsf{topicsim}(b_{new}, b_i))$$

where $\mathsf{topicsim}(b_{new}, b_i)$ is the topic proportion similarity between two bug reports b_{new} and b_i. That is, the highest similarity among $\mathsf{topicsim}(b_{new}, b_i)$ for all b_is will be selected as the topic proportion similarity between b_{new} and group G. Jensen-Shannon divergence, a technique to measure the similarity between two distributions, is used to compute topic proportion similarity. Finally, all duplicate report groups G_js are ranked, and the top-k most similar groups are shown to bug triagers to check for potential duplications for b_{new}.

3.3 BM25F for Textual Similarity Measure

This section describes BM25F, an advanced document similarity function based on weighted word vectors of documents [22, 29]. BM25F considers the retrieval of structured documents which are composed of several fields. Each field in turn corresponds to a word vector, *aka.* a bag of words. Each field in a structure document could be assigned different degrees of importance in the retrieval process. Notationwise, given a set D of N documents, every document d has F fields, and the bag of words in the f^{th} field is denoted by $d[f]$, where $1 \leq f \leq F$. Bug reports are structured doc-

Authorized licensed use limited to: Nanjing University. Downloaded on August 07,2020 at 06:06:37 UTC from IEEE Xplore. Restrictions apply.

uments composed of *summary* and *description* fields; each field corresponds to a bag of words that describe the report.

BM25F computes the similarity between a query and a document based on the common words that are shared between them. BM25F introduces two word importance factors: *global* and *local*. To measure global importance of a word, BM25F computes inverse document frequency (IDF):

$$IDF(t) = log\frac{N}{N_d} \quad (1)$$

In (1), N_d is the number of documents containing the word t. To measure local importance of a word t in a document d, BM25F measures term frequency TF_D which aggregates the local importance of word t for every field in the document d:

$$TF_D(d,t) = \sum_{f=1}^{F} \frac{w_f \times occurrences(d[f],t)}{1 - b_f + \frac{b_f \times length_f}{average\_length_f}} \quad (2)$$

In (2), w_f is the weight of field f; $occurrences(d[f],t)$ is the number of times word t appears in field f of document d; $length_f$ is the size of the bag of words corresponding to $d[f]$; $average\_length_f$ is the average size of the bag of words of the f^{th} fields of all documents in D; and b_f is a parameter ($0 \leq b_f \leq 1$) that controls the normalization of field lengths: $b_f = 0$ and $b_f = 1$ mean no and full normalization, respectively.

Based on the global and local importance, the BM25F score for a document d and a query q is computed as follows:

$$BM25F(d,q) = \sum_{t \in d \cap q} IDF(t) \times \frac{TF_D(d,t)}{c + TF_D(d,t)} \quad (3)$$

In (3), t is the word that appears in both d and q, and c ($c \geq 0$) is a parameter that controls the relative contribution of $TF_D(d,t)$ to the final score.

From Equations (2) and (3), there are a number of free parameters to be specified – these are w_f and b_f for each field f, and c. Thus, with F fields, we need to specify the value of $(2F + 1)$ parameters. These parameters can be automatically determined based on a set of training data following a gradient descent approach as proposed in [30].

3.4 Combination of Topic Model and BM25F

This section describes our technique to combine T-Model and BM25F into DBTM to detect duplicate bug reports. In our model, we have two prediction experts, y_1 is an expert based on topic-based features (T-Model), and y_2 is another expert based on textual features (BM25F). The two experts have different advantages in the prediction of duplicate bug reports. The textual expert (y_2) is stricter in comparison, therefore, it is better in the detection of duplicate bug reports written with the same textual tokens. However, it does not work well with the bug reports that describe the same technical issue with different terms. On the other hand, T-Model can detect the topic similarity between two bug reports even when they are not similar in texts. However, since topic is a way of dimension reduction of text contents, the comparison in topic is less stricter than in texts.

By combining both models, we take advantage of both worlds: textual and topic similarity measurement. To do that, we apply a machine learning technique called Ensemble Averaging, a linear combination of experts [11]. The combined expert is a linear combination of the two experts:

$$y = \alpha_1 * y_1 + \alpha_2 * y_2 \quad (4)$$

where α_1 and α_2 are the parameters to control the significance of experts in estimating duplicate bug reports. They satisfy $\alpha_1 + \alpha_2 = 1$ and are project-specific. If $\alpha_1 = 1$ and $\alpha_2 = 0$, only topic-based expert is used. If $\alpha_1 = 0$ and $\alpha_2 = 1$, only text-based one is used. The optimal values of α_1 and α_2 are learned from the training set (Section 4.3).

4. ALGORITHMS

Let us describe our algorithms for training the parameters of T-Model and those of the combined model DBTM, and our prediction algorithm using the trained parameters.

4.1 Training Algorithm for T-Model

This algorithm aims to estimate T-Model's parameters such as z_b, θ_b, and ϕ_{BR} given the training data from a bug database including the collection of bug reports B, and the set of groups of duplicate bug reports $\{G_j(b)\}$.

We use Gibbs sampling and extend the training algorithm in LDA [6] to support our DBTM. Initially, the parameters z_b and ϕ_{BR} are assigned with random values. The algorithm then iteratively estimates every parameter based on the distribution calculated from other sampled values. The iterative process terminates when the estimated values converge, that is when the sum of the differences between of the current estimated topic distributions and previous estimated ones is smaller than a threshold. In our implementation, the process stops after a number of iterations that is large enough to ensure a small error. The detailed steps are:

1. Estimating the topic assignment for bug reports in B: With each bug report b in B, T-Model estimates the topic assignment $z_b[i]$ for position i. For each topic k in K topics, it estimates the probability that topic k is assigned for position i in document b. Then, it samples a topic based on the probability values of ks. Since each bug report has or does not have duplicate ones, two formulae are needed.

Case 1: When a bug report has no duplicate, the topic assignment estimation follows the Gibbs sampling in LDA [6]:

$$p(z_i = k | z_b[-i], w_b) = \frac{(N_b[-i,k] + \alpha)}{(N_b - 1 + K\alpha)} \frac{(N_{BR,k}[-i,w_i] + \beta)}{(N_{BR,k} - 1 + V\beta)} \quad (5)$$

where $N_b[-i,k]$ is the number of words in b (except for the current position i) that are assigned to topic k; N_b is the total number of words in b; $N_{BR,k}[-i,w_i]$ is the number of words w_i in all bug reports B (except for the current position) that are assigned to topic k; and $N_{BR,k}$ is the number of all words in B that are assigned to topic k.

Case 2: If a bug report b belongs to a duplicate group G_j, they share the same technical issue. Thus, we use the following formula to describe the fact of sharing topic in addition to the local topics of each bug report itself:

$$p(z_i = k | z_b[-i], w_b) = \frac{(N*_b[-i,k] + \alpha)}{(N *_b [-i] + K\alpha)} \frac{(N_{BR,k}[-i,w_i] + \beta)}{(N_{BR,k} - 1 + V\beta)} \quad (6)$$

where $N_{BR,k}[-i,w_i]$ is the number of words w_i in all bug reports in B, except for the current position, that are assigned to k, and $N_{BR,k}$ is the number of words in S describing k.

Comparing to (5), since a duplicate bug report shares the buggy topic with other bug reports in its duplicate group, the proportion θ^* of a topic k described in the bug report includes its local topic proportion θ_b and the topic proportions

Authorized licensed use limited to: Nanjing University. Downloaded on August 07,2020 at 06:06:37 UTC from IEEE Xplore. Restrictions apply.

```
1  // Predict and return a ranked list of groups of duplicate reports
2  function PredictTModel(φ_BR,BugReport b_new, DuplicateGroups G_j)
3    // Estimate topic proportion of new bug report b_new
4    repeat
5      θ'_{b_new} ← θ_{b_new}
6      for (i = 1 to N_b)
7        θ_{b_new} = EstimateZB2(b_new, i) //estimate topic at position i
8      end
9      θ_{b_new}[k] = N_{b_new}[k]/N_{b_new} //estimate topic proportion
10   until (|θ_{b_new} − θ'_{b_new}| <= ϵ)
11   // Calculate topic similarity between bug report b_new and G_j
12   for (DuplicateGroups G_j ∈ B)
13     sim_2(b_new, G_j) = TopicSim(b_new, G_j)
14   end
15   return list (sim_2(b_new, G_j))
16 end
17 // −−−−−− Estimate topic assignment for position i in b −−−−−
18 function EstimateZB2(BugReport b_new,, int i)
19   p(z_{b_new}[i] = k) ← ((N_{b_new}[−i,k]+α)/(N_{b_new}−1+Kα)) ((N_{BR,k}[−i,w_i]+β)/(N_{BR,k}−1+Vβ))
20   z_{b_new}[i] ← sample(p(z_{b_new}[i]))
21 end
22 //Compute topic similarity of b_new and a group of duplicate reports
23 function TopicSim(b_new, G_j)
24   for(BugReports b_i ∈ G_j)
25    TopicSim(b_new, b_i) = 1 − JSDivergence(θ_{b_new}, θ_{b_i})
26   end
27   TopicSim(b_new, G_j) = max_{b_i∈G_j} (TopicSim(b_new, b_i))
28   return TopicSim(b_new, G_j)
29 end
```

Figure 6: Prediction Algorithm

of shared buggy topic θ_{F_j} of the duplicate report group G_j. From (5) and (6), we have $N_b^*[-i,k] = N_b[-i,k]N_{G_j}[k]$ and $n_b^*[-i] = (N_b - 1)N_{G_j}$, in which $N_b[-i,k]$ is the number of words in b (except for the current position i) that are assigned to topic k. N_b is the total number of words in b. $N_{G_j}[k]$ is the total number of positions assigned to topic k in all bug reports in duplicate group G_j and N_{G_j} is the total length of those reports. Note that this equation refers to the impact of the shared topic(s) in the estimation of $\theta_b[k]$ since $\theta_{F_j}[k]$ is reflected (and estimated) via ratio $N_{G_j}[k]/N_{G_j}$.

2. Estimating topic proportion θ_b for a bug report b: Once topic assignments for all positions in b are estimated, the topic proportion $\theta_b[k]$ of topic k in b can be approximated by simply calculating the ratio between the number of words describing the topic k and the length of the document.

3. Estimating word distribution ϕ_{BR}: The last step is to estimate the per-topic word distribution for each word w_i from Voc and topic k. $\phi_k[w_i]$ is approximated by the ratio between the number of times that the word at i-th index in Voc is used to describe topic k and the total number of times that any word is used to describe topic k.

4.2 Prediction Algorithm for T-Model

The goal of this algorithm is to estimate the topic proportion of a newly arrived bug report b_{new} and calculate the topic similarity to other bug reports and duplicate groups. The algorithm uses the trained model from the previous algorithm to estimate the topic proportion of b_{new}, and uses the Jensen-Shannon divergence to calculate the topic similarity between b_{new} and each bug report in all groups of duplicate reports. The similarity sim_1, in combination with BM25F-based similarity sim_2, will be used to estimate how likely b can be a duplicate of the reports in the group G.

```
1  // −−−−−−−−−− Training ensemble weight α_1 −−−−−−−−
2  function TrainAlpha(Reports B, TrainGrps G_train, TestGrps G_test)
3   MAP(G_test, L_pred) ← 0
4   α_1 = 0
5   // Training for T−Model and BM25F models
6   TrainBM25F(B, G_train)
7   TrainTModel(B, G_train)
8   // Compute text and topic similarity of a test report and a group
9   list (sim_1(B_test, G_test) = PredictBM25F(B_test, G_test)
10  list (sim_2(B_test, G_test) = PredictTModel(φ_BR, B_test, G_test)
11  //Estimate α_1
12  for α_1 from 0 to 1
13    increase α_1 by 0.01
14    // Estimate combined similarity , build a ranked list of groups
15   sim(B_test, G_test) =
          α_1 * sim_1(B_test, G_test) + (1 − α_1) * sim_2(B_test, G_test)
16    L_pred = rankedList(sim(B_test, G))
17  return the α_1 value corresponding to the maximum MAP
18 end
```

Figure 7: Ensemble Weight Training Algorithm

The output of the algorithm is a list of potential duplicate bug report groups corresponding to the given bug report.

Figure 6 describes the steps. Lines 4-10 show the estimation step for parameters $z_{b_{new}}$ and $\theta_{b_{new}}$ for new bug report b_{new} (the value of ϕ_{BR} is fixed after training phase and used to estimate z and θ). Since the real duplicate links between b_{new} and bug report groups G are unknown, we use LDA Gibbs sampling equation to estimate the new bug report's topic assignment and topic proportion (Case 1, Section 4.1). The estimation for $z_{b_{new}}$ is described in EstimateZB2 (lines 18-21). In the equation, $N_{b_{new}}[-i,k]$ is the number of words in b_{new} (except the current position i) that are assigned to topic k. $N_{b_{new}}$ is the total number of words in b_{new}. $N_{BR,k}[-i,w_i]$ is the number of words w_i in the collection of bug reports B (except the current position) that are assigned to topic k. $N_{BR,k}$ is the number of words in B assigned to k.

To find the topic similarity between b_{new} and a group of duplicate reports G_j, we calculate TopicSim(b_{new}, G_j) (lines 12-14). TopicSim(b_{new}, G_j) (lines 23-29) is calculated by finding the maximum topic similarity between b_{new} and all bug reports b_is in G_j (line 27). We use the Jensen-Shannon divergence (JSD) to measure the distribution distance between b_{new} and each b_i (line 25). Since JSD is a symmetric measure in [0..1], $1 - JSD$ is topic similarity in [0..1]. Finally, the algorithm returns a list of topic similarity values between b_{new} and all groups of duplicate reports.

4.3 Training for Combined Model DBTM

DBTM is linearly combined from T-Model and BM25F. Thus, we need to determine α_1 and α_2 for calculating the similarity between bug reports and duplicate report groups. Since $\alpha_1 + \alpha_2 = 1$ by definition, DBTM has to learn α_1 only. α_1 can be learned from the training set by using simple cross-validation and a searching algorithm.

Figure 7 shows the training algorithm. Parameters are initialized at lowest possible values (lines 3-4). A training set is used for k-fold cross validation, thus, at each cross validation step, we have $(k-1)$ folds of training duplicate report groups G_{train} and one remaining fold of testing group G_{test}. DBTM first trains T-Model and BM25F model (lines 6-7). The parameters of trained models are used for estimating text similarity levels (line 9) and topic similarity levels (line 10) of a test bug report and a duplicate report group. Those

Authorized licensed use limited to: Nanjing University. Downloaded on August 07,2020 at 06:06:37 UTC from IEEE Xplore. Restrictions apply.

similarity levels are combined into $sim(B_{test}, G_{test})$ via a varying weight α_1 (line 15) with the step of 0.01. The combined similarity values are used to rank the links between bug reports and duplicate report groups (line 16). Those ranked lists of links L_{pred} are used to evaluate a goal function MAP(G_{test}, L_{pred}), which is used to find the optimized value of α_1. The α_1 value corresponding to the highest value for MAP will be returned. The goal function MAP in our algorithm is the mean average precision as proposed in [24].

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

where L_{test} is the real duplicate links in the testing set; L_{pred} is the ranked list of predicted links; $index_i$ is the index where the true duplicate group is retrieved for the i-th query. Since MAP measures how well the algorithm ranks the true links, it can be used as a goal function in training DBTM.

The weights α_1 and α_2 trained from TrainAlpha are used to calculate the combination of text and topic similarity $sim = \alpha_1 * sim_1 + \alpha_2 * sim_2$, where sim_1 and sim_2 are the text and topic similarity between a bug report b_{new} and the duplicate report group G. The higher the combined similarity, the more likely b_{new} is a duplicate of the reports in G.

5. EVALUATION

This section describes our empirical evaluation on DBTM's detection accuracy in comparison with the state-of-the-art approaches, REP [24] and RTM [9]. All experiments were carried out on a computer with CPU AMD Phenom II X4 965 3.0 GHz, 8GB RAM, and Windows 7.

5.1 Data Sets and Feature Extraction

We used the same data sets of bug reports in the open-source projects as in REP [24] (Table 1). Column Time period displays the time period of collected bug reports. Columns Report and Dup show the numbers of bug reports and duplicate ones, respectively. Columns Train and Test show the number of the duplicate bug reports used for training and testing, respectively. The duplication information among bug reports is also available in that data set. The data is used to train T-Model and ensemble weights, and then used to evaluate DBTM's accuracy in detecting the duplication between a bug report and the duplicate bug report groups.

The summary and description of a bug report were merged and considered as a document. It then went through pre-processing such as stemming, and removing grammatical and stop words, and single-occurrence words as in REP [24]. Then, all the words were collected and indexed into a vocabulary. After this phase, a bug report is represented as a vector of the indexes of its words in the vocabulary.

5.2 Evaluation Setting and Metrics

The evaluation setting is the same as in REP [24]. All bug reports were sorted in the chronological order. We divided the data set into two sets. The training set includes the first M reports in the repository, of which 200 reports are duplicates. It was used to train the parameters for T-Model, BM25F, and DBTM. The remaining reports were used for testing. At each execution, we ran DBTM through the testing reports in the chronological order. When it determines a duplicate report b, it returns the list of top-k potential duplicate report groups. If a true duplicate report group G is found in the top-k list, we count it as a hit. We then added b to that group for later training. The top-k accuracy (i.e. recall rate) is measured by the ratio of the number of hits over the total number of considered bug reports.

Table 1: Statistics of All Bug Report Data

Project	Time period	Report	Dup	Train	Test
OpenOffice	01/01/2008 - 12/21/2010	31,138	3,371	200	3,171
Mozilla	01/01/2010 - 12/31/2010	75,653	6,925	200	6,725
Eclipse	01/01/2008 - 12/31/2008	45,234	3,080	200	2,880

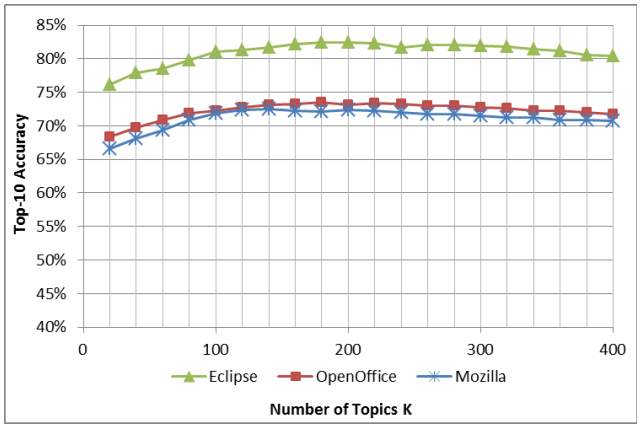

Figure 8: Accuracy with Varied Numbers of Topics

5.3 Sensitivity Analysis

In the first experiment, we evaluated the sensitivity of DBTM's accuracy with respect to different numbers of topics K. We ran DBTM on Eclipse data set as K was varied from 20 to 400 with the step of 20, and then measured top-10 detection accuracy. Figure 8 shows the result. The shapes of the graphs for three systems are consistent. That is, as K is small (K<60), accuracy is low. This is reasonable because the number of features for bug reports is too small to distinguish their technical functions, thus, there are many documents classified into the same topic group even though they contain other technical topics. When the number of topics increases, accuracy increases as well and becomes stable at some ranges. The stable ranges are slightly different for different projects, however, they are large: K=[140-320] for Eclipse, K=[120-300] for OpenOffice, and K=[100-240] for Mozilla. This suggests that in any value of K in this range for each project gives high, stable accuracy. The reason might be because the number of topics in these ranges reflect well the numbers of technical issues in those bug reports. However, as K is larger (K>380), accuracy starts decreasing because the nuanced topics appear and topics may begin to overlap semantically with each other. It causes a document to have many topics with similar proportions. This overfitting problem degrades accuracy.

5.4 Accuracy Comparison

In this experiment, we aimed to evaluate how topic-based features in our topic model T-Model, in combination with BM25F, can help to detect duplicate bug reports. We also compared our combined model DBTM with REP [24]. The parameter K of DBTM in this experiment was selected after fine-tuning for best results as in the previous experiment.

Authorized licensed use limited to: Nanjing University. Downloaded on August 07,2020 at 06:06:37 UTC from IEEE Xplore. Restrictions apply.

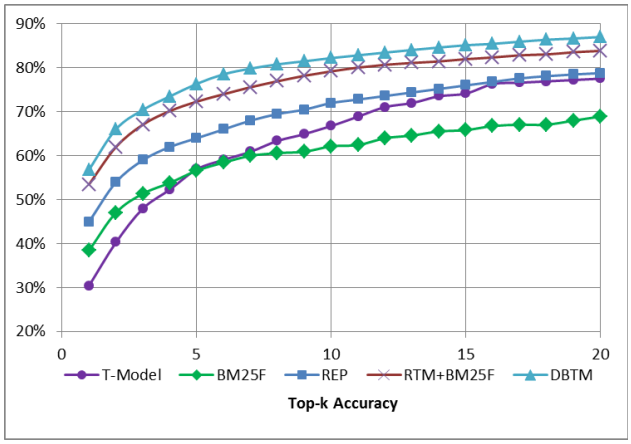

Figure 9: Accuracy Comparison in Eclipse

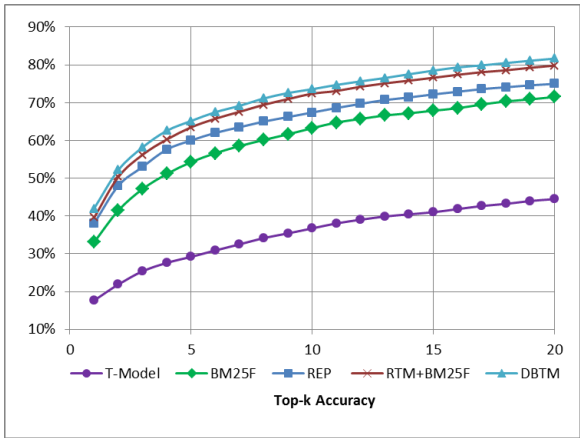

Figure 10: Accuracy Comparison in OpenOffice

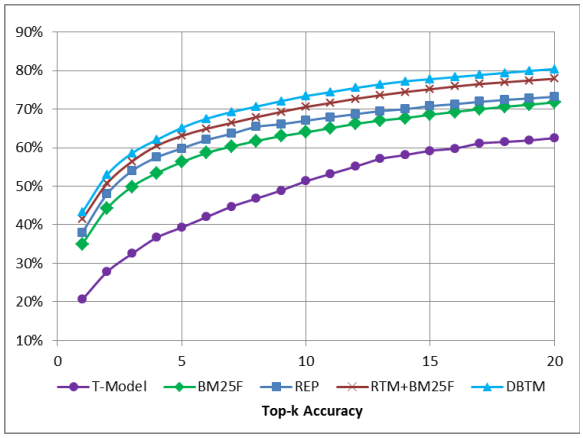

Figure 11: Accuracy Comparison in Mozilla

Figure 9 displays the accuracy result of DBTM in comparison with REP on Eclipse data set. We used REP's result from [24] because the same data sets and experiment setting were used in this study. As shown, DBTM achieves very high accuracy in detecting bug reports. For a new bug report, in 57% of the detection cases, DBTM can correctly detect the duplication (if any) with just a single recommended bug report (i.e. the master report of the suggested group). Within a list of top-5 resulting bug reports, it correctly detects the duplication of a given report in 76% of the cases. With a list of 10 reports, it can correctly detect in 82% of the cases. In comparison, DBTM achieves higher accuracy from 10%-13% for the resulting lists of top 1-10 bug reports. That is, it can relatively improve REP by up to 20% in accuracy.

We also compared the performance of two individual components in DBTM. We implemented BM25F for comparison. As seen, the IR approach BM25F generally achieves higher accuracy than T-Model alone (except for top-5 accuracy and above for Eclipse). Examining this case, we see that topic model tends to group the bug reports with the same topics, but not necessarily duplicates of one another. REP [24], an extension from BM25F, outperformed both topic model and BM25F. Those features such as non-textual fields (e.g. product, component, and version) clearly help improve the performance of BM25F. However, because DBTM achieves 10%-13% higher than REP, the topic-based features from T-Model help improve further the performance of BM25F than those non-textual fields. We found that in several cases, REP was not able to detect the duplications of bug reports whose texts are not similar, while they can be identified by DBTM via topic features. That is, DBTM takes the best of both worlds: topic modeling and information retrieval.

The results are also consistent in other data sets: OpenOffice and Mozilla. Figures 10 and 11 display the accuracy results on OpenOffice and Mozilla data sets, respectively. DBTM consistently achieves very high levels of accuracy (42-43% for top-1, 65-67% for top-5, and 73-74% for top-10 accuracy). In comparison with REP [24], DBTM consistently improves over REP with higher accuracy from 4%-6.5% for OpenOffice and 5%-7% for Mozilla (i.e. 10-12% relatively).

To compare DBTM with a state-of-the-art topic model, RTM [9], we implemented the combined model of RTM and BM25F. RTM is a topic model extended from LDA by modeling the presence of the observed links between documents. As seen in Figures 9-11, our DBTM outperformed RTM+BM25F from 4-7% (i.e. 5-11% relatively). This result shows that combining topic modeling with IR can achieve better results than individual techniques. Moreover, our T-Model is more specialized toward duplicate bug reports and performed better than RTM. This is reasonable. First, in RTM [9], the presence of a link between two documents depends on the similarity of their respective topic proportions. Two duplicate bug reports do not necessarily have similar topic proportions (Section 2). They might contain more of their own topics. Second, in practice, there are often more than two duplicate reports in a group. RTM must be trained for each pair of those duplicate reports and it aims to find the common topic structure among *the document pair*, rather than the shared buggy topic(s) among *all duplicate reports* in a group. DBTM can naturally find the shared topic(s) and does not focus on individual pairs. For these data sets, we found that there are many groups with two duplicate reports. Thus, the results for RTM might get worse in other subject systems if groups contain more than two reports.

5.5 Time Efficiency

Figure 12 shows DBTM's time efficiency result. The size of a project is the total of the number of bug reports and

Authorized licensed use limited to: Nanjing University. Downloaded on August 07,2020 at 06:06:37 UTC from IEEE Xplore. Restrictions apply.

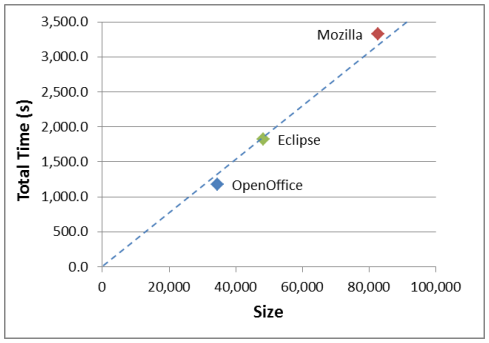

Figure 12: Time Efficiency

the number of duplicate bug report groups in each data set because training/predicting considers both bug reports and duplicate bug report groups. The sizes are 34,509, 48,314, and 82,578 for OpenOffice, Eclipse, and Mozilla respectively. The total training and predicting time for those projects are 1,174.8s, 1,819s, and 3,323.7s respectively. As seen, the time is about linear to a project's size, e.g. $\frac{time(Eclipse)}{size(Eclipse)} \approx \frac{time(Mozilla)}{size(Mozilla)}$. Importantly, DBTM is highly efficient. For a large project like Mozilla, it took about 5 minutes for training (which could be run in background). For predicting, on average, prediction time for one bug report are just 0.031s, 0.035s, and 0.041s for OpenOffice, Eclipse, and Mozilla, respectively. In brief, DBTM is scalable and efficient to be used interactively in detecting duplicate bug reports.

Interesting Case Studies Figure 13 shows two duplicate bug reports detected by DBTM. Except the terms NPE (NullPointerException) and StructuredViewer, which are popular and common in the project, the two reports contain several different terms because the reporters found the bug in two different usage scenarios. That leads to different exception traces: one involving image updating, and another on widget selection. We noticed that when running BM25F model by itself, bug report #225169 is ranked 8^{th} in the list that could be duplicate of bug report #225337 due to the dissimilarity in texts. However, after extracting topics via the co-occurrences of other terms such as startup, first time, RSE perspective, wizard, etc in the previous duplicate reports (e.g. from bug report #218304, not shown), DBTM ranked it at the highest position and detected them as duplicate ones.

Threats to Validity We evaluated only on three open-source projects. Different projects might have different quality of bug reports. However, Eclipse, Mozilla and OpenOffice are long-lasting projects and were used in prior research. They are sufficiently representative for our comparison. We also should validate our method on commercial projects.

6. RELATED WORK

Several approaches have been introduced to support the automatic detection of duplicate bug reports. Earlier approaches have applied traditional information retrieval (IR) methods to this problem [14, 23]. Hiew *et al.* [14] use Vector Space Model (VSM) by modeling a bug report as a vector of textual features computed from Tf-Idf term weighting measurement. Vector similarity is used to measure the similarity among bug reports. Runeson *et al.*'s [23] combine VSM

Bug Report #225169
Summary: Get NPE when startup RSE on a new workspace
Description:
Using Eclipse M5 driver and RSE I20080401-0935 build. Start eclipse on a new workspace, and switch to RSE perspective. I could see the following error in the log. But otherwise, things are normal.
java.lang.NullPointerException at
org.eclipse.....getImageDescriptor(SystemView...java:123)
...
at org.eclipse....doUpdateItem(AbstractTreeViewer.java:1010)
at org.eclipse....doUpdateItem(SafeTreeViewer.java:79)
at org.eclipse....run(StructuredViewer.java:466)...

Bug Report #225337
Summary: NPE when selecting linux connection in wizard for the first time
Description:
After starting an eclipse for the first time, when I went select Linux in the new connection wizard, I hit this exception. When I tried again a few times later, I wasn't able to hit it.
java.lang.NullPointerException at
org.eclipse....getAdditionalWizardPages(RSEDefault...:404)
...
at org.eclipse....updateSelection(StructuredViewer.java:2062)
at org.eclipse....handleSelect(StructuredViewer.java:1138)
at org.eclipse....widgetSelected(StructuredViewer.java:1168)...

Figure 13: Duplicate Bug Reports in Eclipse

with simple natural language processing (NLP) to improve further. In addition to NLP, Wang *et al.* [26] also take advantage of the information on execution traces relevant to the bugs in the bug reports. However, such information might not always be available as it is reported that only a small percentage of bug reports (0.83%) contain execution traces [25]. Comparing to those approaches, DBTM operates at a higher level of abstraction by also comparing the underlying technical topics in the bug reports.

Another line of approach to this problem is machine learning (ML). Jalbert and Weimer [16] developed a binary classifier model by performing a linear regression over *textual* features of bug reports computed from the frequencies of terms in bug reports. To make a binary classifier, they specify an output value cutoff over such features that distinguishes between duplication and non-duplication. Support Vector Machine (SVM) was used by Sun *et al.* [25] in which to train an SVM classifier, all pairs of duplicate bug reports are formed and considered as the positive samples, and all other pairs of non-duplicate bug reports are used as the negative ones. For prediction, a new report is paired with all existing reports. Then, each pair is fed into the trained SVM model and classified as positive or negative. Positive pairs imply duplicate reports. The key limitation of those aforementioned ML approaches is low time efficiency.

To overcome that, recent work by Sun *et al.* [24] introduced REP, a novel IR technique that extends BM25F to consider the long bug reports and the meta-data such as the reported product, component, and version. They showed that REP outperformed the state-of-the-art ML approaches in both accuracy and efficiency. In this work, we combine BM25F with our novel topic model, T-Model, to address the cases where duplicate reports have different terms for the same technical issue. To our knowledge, DBTM is the first work in which topic-based features are used with IR to support the detection of duplicate bug reports.

Authorized licensed use limited to: Nanjing University. Downloaded on August 07,2020 at 06:06:37 UTC from IEEE Xplore. Restrictions apply.

Our prior work, BugScout [20], has applied topic modeling in the bug localization problem in which for a new bug report, it provides a list of potential buggy files for the reported issue. DBTM is also based on topic modeling, however, it has significant differences from BugScout. First, while DBTM has one monolithic model with one single vocabulary set, BugScout correlates the terms in two separate sub-models to represent two different types of documents with two vocabulary sets (source code in a programming language, and bug reports in a natural language). Second, in BugScout, the buggy source files, which are observable, affect the topic assignment of the corresponding bug report. In DBTM, it is more challenging because the buggy topics (i.e. the technical issues) are latent and DBTM correlates the terms in the duplicate bug reports that share the technical issue(s) without determining it.

Other researchers focus generally on bug reports. It is suggested that duplicate reports complement to one another to help in bug fixing [4]. Bettenburg *et al.* [3] analyzed information mismatch between what developers need and what users supply to determine good properties in bug reports. Structural information from bug reports has been shown to be useful [5, 18]. Other researchers studied bug reports based on their types, quality, severity [15, 3, 15, 1, 21, 10, 19, 2, 27, 12]. From bug reports, prediction tools [28, 17] can tell whether a bug could be resolved with certain fixing time. Approaches for automatic bug triaging include [1, 7, 8].

7. CONCLUSIONS

This paper introduces DBTM, a duplicate bug report detection approach that considers not only IR-based features but also topic-based features. DBTM models each bug report as a textual document describing one or more technical issues, and models duplicate bug reports as the ones on the same technical issue. Trained with historical data including identified duplicate reports, DBTM can learn sets of different terms describing the same technical issues and detect other not-yet-identified duplicate ones. Our empirical evaluation on real-world systems shows that DBTM can improve the state-of-the-art approaches by up to 20% in accuracy.

8. ACKNOWLEDGMENTS

This project is funded in part by US National Science Foundation (NSF) CCF-1018600 grant.

9. REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In ICSE '06, pages 361–370. ACM, 2006.

[2] N. Bettenburg, S. Just, A. Schröter, C. Weiß, R. Premraj, and T. Zimmermann. Quality of bug reports in Eclipse. In ETX'07, pp. 21–25. ACM, 2007.

[3] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In FSE'08, pages 308–318. ACM, 2008.

[4] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Duplicate bug reports considered harmful... really? In ICSM'08, pages 337–345. IEEE CS, 2008.

[5] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Extracting structural information from bug reports. In MSR'08, pages 27–30. ACM, 2008.

[6] D. M. Blei, A. Y. Ng, M. I. Jordan. Latent Dirichlet Allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003.

[7] G. Canfora, L. Cerulo. How software repositories can help in resolving a new change request. In WESRE'05.

[8] G. Canfora and L. Cerulo. Supporting change request assignment in open source development. In SAC'06.

[9] J. Chang and D. Blei. Relational topic models for document networks. In *AIStats*, 2009.

[10] D. Cubranic and G. Murphy. Automatic bug triage using text categorization. In SEKE'04. KSI, 2004.

[11] Ensemble averaging. http://en.wikipedia.org/wiki/Ensemble_averaging.

[12] M. Fischer, M. Pinzger, and H. Gall. Analyzing and relating bug report data for feature tracking. In WCRE'03. IEEE CS, 2003.

[13] Gibbs sampling. wikipedia.org/wiki/Gibbs_sampling

[14] L. Hiew. Assisted detection of duplicate bug reports. Master's thesis, University of British Columbia, 2006.

[15] P. Hooimeijer and W. Weimer. Modeling bug report quality. In ASE'07, pages 34–43. ACM, 2007.

[16] N. Jalbert and W. Weimer. Automated duplicate detection for bug tracking systems. In *Int. Conf. on Dependable Systems and Networks*, pp. 52–61. 2008.

[17] S. Kim and E. J. Whitehead, Jr. How long did it take to fix bugs? In MSR'06, pages 173–174. ACM, 2006.

[18] A. J. Ko, B. A. Myers, and D. H. Chau. A linguistic analysis of how people describe software problems. In VLHCC'06, pages 127–134. IEEE CS, 2006.

[19] T. Menzies and A. Marcus. Automated severity assessment of software defect reports. In ICSM'08.

[20] A. T. Nguyen, T. T. Nguyen, J. Al-Kofahi, H. V. Nguyen, and T. N. Nguyen. A Topic-based Approach for Narrowing the Search Space of Buggy Files from a Bug Report. In ASE'11, pp. 263-272. IEEE CS, 2011.

[21] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang. Automated support for classifying software failure reports. In ICSE'03.

[22] S. Robertson, H. Zaragoza, and M. Taylor. Simple BM25 extension to multiple weighted fields. In CIKM'04, pages 42–49. ACM, 2004.

[23] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In ICSE'07. IEEE CS, 2007.

[24] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang. Towards more accurate retrieval of duplicate bug reports. In ASE'11, pages 253–262. IEEE CS, 2011.

[25] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo. A discriminative model approach for accurate duplicate bug report retrieval. In ICSE'10. ACM, 2010.

[26] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In ICSE'08, pages 461–470. ACM, 2008.

[27] W. Weimer. Patches as better bug reports. In GPCE'06, pages 181–190. ACM, 2006.

[28] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In MSR'07.

[29] H. Zaragoza, N. Craswell, M. J. Taylor, S. Saria, and S. Robertson, Microsoft Cambridge at TREC 13: Web and HARD tracks, in *TREC*, 2004.

[30] M. Taylor, H. Zaragoza, N. Craswell, S. Robertson, and C. Burges, Optimisation methods for ranking functions with multiple parameters, In CIKM'06.

Authorized licensed use limited to: Nanjing University. Downloaded on August 07,2020 at 06:06:37 UTC from IEEE Xplore. Restrictions apply.