EXPSOL: Recommending Online Threads for Exception-related Bug Reports

Xiaoning Liu, Beijun Shen, Hao Zhong, Jiangang Zhu
School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China
{donlxn, bjshen, zhonghao, jszjgtws}@sjtu.edu.cn

***Abstract*—An exception-related bug is a kind of program bug which causes exceptions. During software maintenance, when programmers repair exception-related bugs, they typically analyze thrown exceptions to understand the root causes of such bugs. When encountering unfamiliar thrown exceptions, programmers often refer to online forum threads (*e.g.* StackOverflow) to understand how to fix them. Although some general search engines are available and some research tools are proposed, they are insufficient to recommend threads for exception-related bugs from large-scale online resources. In this paper, we propose an approach, named EXPSOL, which recommends online threads as solutions for a newly reported exception-related bug with a model trained by support vector machines. We conduct two evaluations on thousands of threads from StackOverflow and fixed issues from GitHub. The results of our first evaluation show the significance of our internal features and highlight the importance of integrating different features. The results of our second evaluation show that, EXPSOL performs better in mean average precision, mean reciprocal rank and recall than those of the Google search engine, the internal search engine of StackOverflow, and other existing approaches.**

Keywords*—*Recommendation, Exception-related Bug, StackOverflow, GitHub.

I. Introduction

An exception is an abnormal event which occurs during the routine execution of a program, and a thrown exception often indicates a bug to be fixed [17]. Most modern programming languages (*e.g.*, Java and C#) implement exception handling mechanisms [5]. As thrown exceptions often indicate bugs, during software maintenance, many exception-related bugs are reported through issue trackers (*e.g.*, GitHub[1]), and it can take much effort to fix such reported bugs [26]. When programmers encounter reported bug that is related to an unfamiliar exception, they often search online forums (*e.g.*, StackOverflow[2]) to understand such bugs. Although bug reports and forum threads seem to be irrelevant, we find many connections between them. For example, we find that many bug reports have URLs to forum threads, and Section II presents such an example. The connections show the usefulness of forum threads, when programmers fix exception-related bugs. Furthermore, it can be feasible to train a model from known connections, and such a model can recommend threads for a newly reported exception-related bug. The recommendation can benefit the repair process of exception-related bugs.

[1]http://www.github.com
[2]http://stackoverflow.com

Although returned online threads can be beneficial, it is often challenging to find useful ones, since there are quite an amount of online threads. In literature, researchers [11], [18], [22] proposed approaches that recommend online resource to assist programming tasks, but these approaches are designed for other purposes than recommending online threads to exception-related bug reports. Rahman *et al.* [24] integrate StackOverflow with Eclipse, and recommend threads when the current code throws errors or exceptions. However, their approach cannot recommend proper threads for bug reports, since bug reports have quite different contents from the error messages from Eclipse.

Indeed, bug reports have specific structures which require more advanced analysis. For example, in a bug report, a thrown exception may have inheritance relationships with other exceptions or classes. Although such details are useful to locate online threads, to the best of our knowledge, previous approaches do not fully use such information.

After analyzing thousands of bug reports and forum threads, we identify three challenges in recommending online threads for exception-related bugs: (1) as forum threads and bug reports are written in the mixed format of natural languages, code samples, and stack traces, it is challenging to determine proper features for the recommendation; (2) as forum threads and bug reports are extracted from two different sources, it is challenging to make the connection between threads and reported bugs; and (3) it is challenging to recommend proper online threads for a newly exception-related bug report, since there are millions of online threads.

In this paper, we propose an exception-related bug solution recommender, EXPSOL, and identify three types of potentially useful features. Based on the features, we train a model that is based on support vector machines. Our trained model captures the similarity between bugs and corresponding useful forum threads, and thus is able to recommend useful threads for a newly reported bug. To show the effectiveness of our approach, we select GitHub and StackOverflow as our subjects in our evaluation, since they are popular and widely used as research subjects in existing studies [2], [6], [13], [27]. This paper makes these major contributions:

- An identity linkage algorithm for exception-related resources by leveraging structured exception messages and relationships. Based on the algorithm, an exception tree is constructed to link GitHub issues with threads from StackOverflow.

- A novel approach, called EXPSOL, which combines three kinds of features with a semi-supervised learning model to recommend online threads to reported exception-related bugs.
- An evaluation on thousands of bug reports and forum threads from GitHub and StackOverflow. Our evaluation shows that EXPSOL achieves better results than three compared approaches.

The rest of this paper is organized as follows. Section II presents a motivating example. Section III describes our approach. Section IV performs evaluations. Section V discusses the validity of our work. We review the related work in Section VI and summarize this paper in Section VII.

II. Motivating Example

Suppose that a programmer named Mary, encounters a reported bug that relates to `JsonMappingException`. The bug throws the stack trace as follow:

```
com.fasterxml.jackson.databind.JsonMappingException: ...
 at ...StringSerializer.serialize(StringSerializer.java:49)
 at ...MapSerializer.serializeFieldsUsing(...)
    ...
```

The following code sample triggers the exception:

```
public void testStringArrayInMap() throws Exception{
      Properties map = new Properties();
      map.put("names", new String[]"david", "filip", "ivan");
      System.out.println(MAPPER.writeValueAsString(map));
}
```

The reporter of the bug presents the following description:

> There is possible bug introduced in tags/jackson-databind-2.6.0: I tested this on Ubuntu Linux and Windows 7 with java 8; Simple workaround for this problem is to use java.util.HashMap instead of java.util.Properties.

If Mary is unfamiliar with the thrown exception, she may search for online threads to understand its causes and how to fix the thrown exception. In particular, she can break the full name of the exception into several keywords then search them with Google.

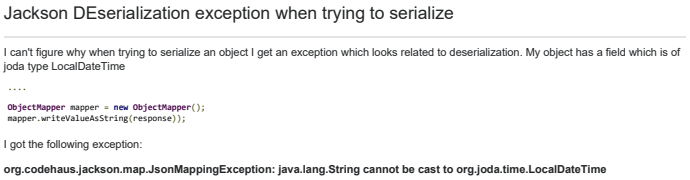
When other programmers have different opinions in fixing an exception-related bug, they can also refer to threads on online forums. For example, Figure 1(a) shows a fixed bug[3] in Github. The bug throws the same exception as the above newly reported bug. When programmers discussed how to fix the bug, as shown in Figure 1(b), a programmer, called JacksonTi, pointed out the cause of the exception, and referred to a thread[4] on StackOverflow. Figure 1(c) shows the thread. The answers to the thread in Figure 1(d) say that the exception may be caused by incorrect type conversions. If Mary knows the link, she can fix the newly reported bug more effectively.

However, such links are hidden. Existing search engines are insufficient to discover such links, since they typically use only the lexical similarity between selected keywords and online threads. Even if such links are discovered, it is challenging to recommend threads for a newly reported bug, based on such links. Oftentimes, Mary has to manually identify useful

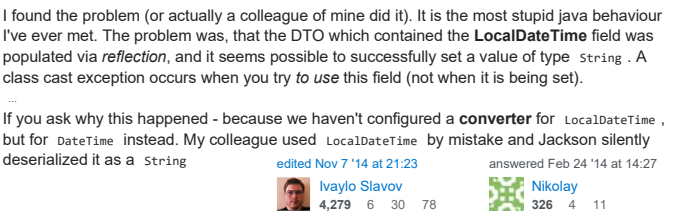
[3]https://github.com/FasterXML/jackson-databind/issues/1174
[4]https://stackoverflow.com/questions/21988512

(a) A fixed exception-related bug (FasterXML1174) in the GitHub

(b) A URL in the discussion of FasterXML1174



(c) The related StackOverflow thread for FasterXML1174



(d) The useful answer for FasterXML1174

Fig. 1. An example link between fixed bug and forum thread

ones from many returned results, although it is tedious, and determined by selected keywords.

EXPSOL extracts the links between fixed bugs and corresponding threads, and trains a model based on their key features (Section III). For a reported bug, it automatically recommends its useful threads from StackOverflow. For example, when Mary is fixing the newly reported bug, EXPSOL can recommend the useful thread in Figure 1(c), since its trained model identifies the semantic similarity between fixed bugs and corresponding threads.

III. Approach

In this section, we first define our research problem (Section III-A). We propose an approach, called EXPSOL, that recommends forum threads to a bug report. Figure 2 shows the overview of EXPSOL. It consists of four key components: exception tree construction (Section III-B), exception tree tagging (Section III-C), feature extraction (Section III-D), and SVM-based recommendation (Section III-E).

A. Problem Definition

In our definition, we use R as a unified format to denote online resource such as threads on StackOverflow, and call

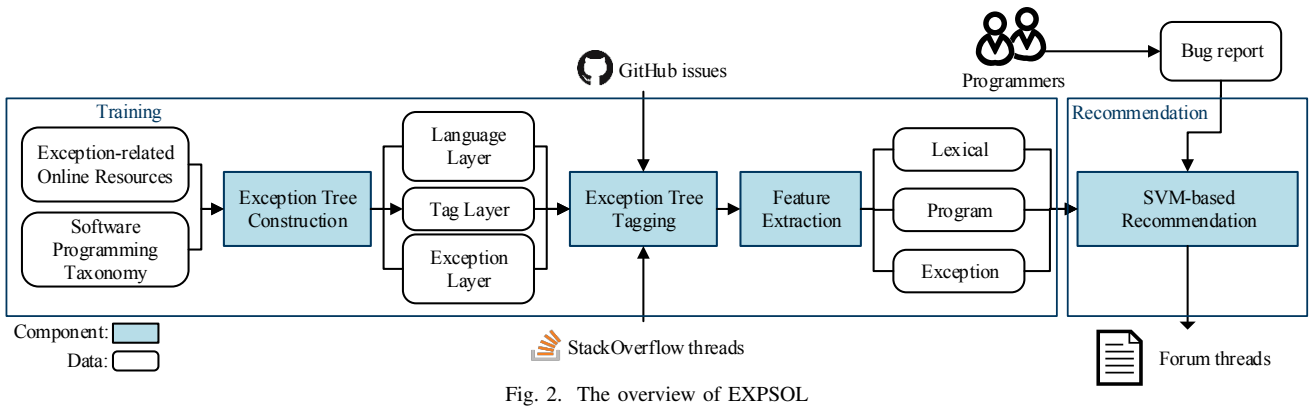


Fig. 2. The overview of EXPSOL

each item $r \in R$ a resource item. Here, r is a seven-tuple $\langle i_r, b_r, T_r, ST_r, CS_r, EP_r, P_r \rangle$, where i_r is its title; b_r is its descriptions; T_r is its tags; ST_r is its stack traces; CS_r is its code samples; EP_r is its extracted exception-related keywords; and P_r is its replies.

We define a newly reported exception-related bug c as a five-tuple $\langle l_c, ST_c, CS_c, EP_c, ED_c \rangle$, where l_c is the name of its thrown exception; ST_c is its stack trace; CS_c is its code samples; EP_c is extracted exception-related keywords; and ED_c is its descriptions.

We then define the recommendation problem as follow: for a $c \in C$, the recommendation compares c with each resource item $r \in R$, and its goal is to find a function $f : c \times R \mapsto \{0, 1\}$. If $(c, r) \in C \times R \wedge f(c, r) = 1$, it recommends r to c, others not. We thus reduce our research problem as a binary classification problem.

B. Exception Tree Construction

This component takes exception-related online resources and a software programming taxonomy [31] as its inputs, and builds an exception tree. The tree has three layers such as the languages layer, the tag layer, and the exception layer.

1) The Language Layer: The first layer of the exception tree is the language layer. We support programming languages which currently implement exception handling mechanisms in this layer. Each node in this layer denotes a programming language, and the *others* node denotes other languages for the future extension.

2) The Tag Layer: Most threads in StackOverflow and most issues in GitHub have tags. Although such tags are useful to classify online resources, they are not clearly defined. In our previous work [31], we have built a software programming taxonomy. The taxonomy is available from Datahub website[5], and our website provides a query interface[6]. As the taxonomy is extracted from StackOverflow, it describes the hierarchical semantic structures of tags on StackOverflow. Based on the taxonomy, for each language, we construct the tag layer. The major challenge lies in identifying children nodes from the taxonomy. Algorithm 1 defines a recursive function with depth first traversal.

[5]http://datahub.io/dataset/software-zhishi-schema

[6]http://swenet.me

In the algorithm, we assume that there is a set of n tags in the taxonomy $T = \{t_1, \ldots, t_n\}$. The hyponym information is summarized in an $n \times n$ matrix H, where $H_{ij} = 1$ denotes that t_i is a hyponym of t_j, $Hij = 0$ otherwise. We use $N = \{n_1, \ldots, n_n\}$ to denote nodes of the exception tree, and E to denote its edges. An edge $e_{ij} \in E$ denotes that n_i is the children of n_j. We build the tag layer from selected tags in the taxonomy T^*, and denote the nodes of the top two layers as N^*. We then match tags with nodes by means of the same keywords or synonyms[7], and define the matching function $\zeta : T^* \mapsto N_T^*$ and $\eta : N_T^* \mapsto T^*$.

3) The Exception Layer: This layer consists of the structure of exceptions. We could extract the structure of individual thrown exceptions. For example, from *java.lang.NullPointerException*, we extract three keywords such as *java*, *lang* and *NullPointerException*, and add the corresponding nodes to the exception layer.

We further synthesize the structure of different exceptions, and define a ξ function which builds the connections between the tag layer and the exception layer. Algorithm 2 shows the construction process. We locate exception tree tag layer nodes which match with tags from the inputted resource.

Different exceptions can share the same prefix. For example, *jsp.JspTagException* and *jsp.SkipPageException* share the same *jsp* prefix. When this happens, we put the *JspTagException* node and the *SkipPageException* node under the same parent *jsp* node.

C. Exception Tree Tagging

This component builds linkages between exception-related threads and issues. In particular, it adds those online resources as leaf nodes to proper locations of the exception tree. As shown in the last line of Algorithm 2, for each resource item r, it links r with the exception nodes that are newly added in exception tree E_{EP_r}.

There are two special cases: (1) One thread or issue can be linked to multiple nodes. To handle the case, Algorithm 2 links a thread or an issue to the common parent node of all the related nodes. (2) Some threads and issues cannot be linked to any node. For such resource items, Algorithm 2 links them to the *others* node.

[7]The synonyms relations are in http://stackoverflow.com/tags/synonyms

Algorithm 1 Identifying Child Nodes in Tag Layer

Input:
A language or tag layer node n_j in exception tree
Exception tree nodes set N
Exception tree edges set E
Taxonomy T and hyponym matrix H
Output:
Updated tree nodes set N and edges set E

$t_j = \eta(n_j)$
while $t_i \in T$ **and** $H_{ij} = 1$ **do**
  $n_i = \zeta(t_i)$
  $N = N + \{n_i\}$
  $E = N + \{e_{ij}\}$
  $Algorithm\ 1(n_i)$
end while

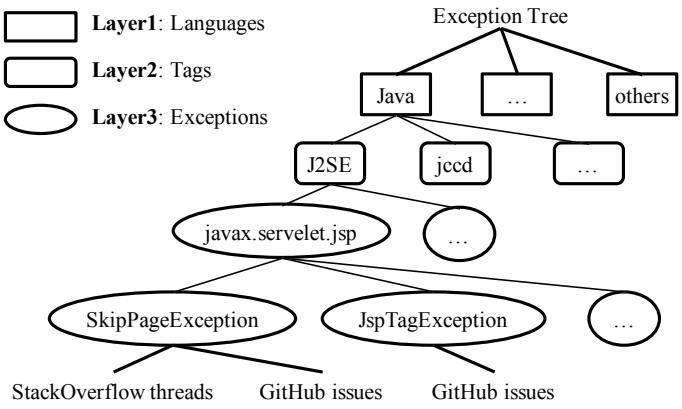


Fig. 3. Exception tree

D. Feature Extraction

EXPSOL uses three types of features: lexical feature, program features and exception tree features. We define the similarity metrics between a fixed bug b and an online thread t as follows:

1) Lexical Feature (LF): The lexical similarity is as follow:

$$sim_{LF}(b,t) = \frac{\vec{b} \cdot \vec{t}}{\|\vec{b}\| \cdot \|\vec{t}\|} \tag{1}$$

In the above equation, $\vec{b}$ and $\vec{t}$ denote the corresponding term vector of b and t. Before calculation, we remove stop words and links from b and t. We also stem the words which can significantly improve results based on lexical similarity.

2) Program Features (PF): We define three similarity metrics for stack traces, code samples, and APIs.

1. Stack traces provide useful details to debug exception-related bugs. Based on the code similarity in literature [25], we define the similarity between track stacks:

$$sim_{PF1}(b,t) = \frac{1}{ham(b_{stack}, t_{stack})} \tag{2}$$

where b_{stack} and t_{stack} denote their stack traces, and ham denotes their Hamming distance.

Algorithm 2 Identifying Exception Layer Nodes and Tagging of Online Resource

Input:
A StackOverflow question or GitHub issue r
Exception tree nodes set N and edges set E
Output:
Updated exception tree nodes set N and edges set E

Extract resource tags set T_r from r
Extract exceptions keywords set EP_r from r
Locate tag layer nodes N_r^* which match with T_r
if N_r^* is $empty$ **then**
  $N_r^* = \{n_{others}\}$
end if
while $n_i \in N_r^*$ **do**
  if n_i is parent of any node in $N_r^* - \{n_i\}$ **then**
    $N_r^* = N_r^* - \{n_i\}$
  end if
end while
$N = N + N_{EP_r}$
$E = E + \xi(N_{EP_r}) + \xi(N_{EP_r}, N_r^*)$
Link r with N_{EP_r}

2. Code samples can be provided to illustrate thrown exceptions. We define the similarity between code samples:

$$sim_{PF2}(b,t) = \frac{1}{ham(b_{code}, t_{code})} \tag{3}$$

where b_{code} and t_{code} denote their code samples.

3. APIs are important in programming tasks, and wrong APIs can lead to bugs [30]. With the help of regular expression, we extract the APIs, and define their similarity:

$$sim_{PF3}(b,t) = \frac{|b_{API} \cap t_{API}|}{|b_{API} \cup t_{API}|} \tag{4}$$

where b_{API} and t_{API} denote the extracted APIs from b and t, respectively.

3) Exception Tree Features (ETF): After the exception tree construction and tagging process, all the exception-related threads and issues become leaf nodes of the exception tree. Based on the tree structure, we define the following two similarity metrics.

1. Average Exception Tree Path Score. In a tree, two high correlation nodes often have a short path, and the nodes between them have small degrees. If the path from node n_j to n_k in the exception tree is $P = \{n_j, \cdots, n_k\}$, we define the weight of the path:

$$\Gamma(P) = \sigma(n_k) \prod_{i=j}^{k-1} \frac{\sigma(n_i)}{|degree(n_i)|^{\sigma}} \tag{5}$$

where

$$\sigma(n) = \begin{cases} \alpha, n \in N_{layer1} \\ \beta, n \in N_{layer2} \\ \sigma, n \in N_{layer3} \end{cases}$$

In the above equation, α, β and σ are parameters describing influences of three different layer nodes. We thus define the similarity metric as average exception tree path score for b and c as follow:

$$sim_{ETF1} = \frac{\sum_{n_i \in b_N} \sum_{n_j \in t_N} \Gamma(P_{ij})}{\sum_{n_i \in b_N} \sum_{n_j \in t_N} (1)} \quad (6)$$

where b_N and t_N denote the neighbors of b and t, respectively.

2. Average Exception Tree Altitude Difference. As two nodes with a smaller height difference can share more in common, we define the similarity metric to measure the altitude difference of b and t as follow:

$$sim_{ETF2} = \frac{\sum_{n_i \in b_N} \sum_{n_j \in t_N} HeightDiff(n_i, n_j)}{\sum_{n_i \in b_N} \sum_{n_j \in t_N} (1)} \quad (7)$$

where $HeightDiff$ calculates their altitude difference.

E. SVM based Recommendation

We reduce the recommendation problem to a binary-class classification problem. In machine learning, support vector machines (SVMs) [16] are supervised learning models, and are widely used in classification. As SVM is one of the best classifiers, we use it to solve our problem. As mentioned in Section III-A, we do not have to calculate one exception context c with all of the forum threads, which is very time-consuming. Instead, we propose a filtering algorithm that is based on the exception tree. As a result, EXPSOL needs to compare only a smaller candidate set when it recommends online threads.

1) Model Training: In SVM, the training data includes a set of pairs $\{\langle \vec{x}_1, l_1 \rangle, \ldots, \langle \vec{x}_n, l_n \rangle\}$, where $\vec{x}$ denotes a similarity vector, and l denotes its label. A similarity vector $\vec{x}_i$ includes all the similarity values between a fixed exception-related bug c_i and an online thread t_i. As our trained model identifies two types of online threads, we define two types of labels. In particular, a positive label denotes that an online thread is useful for a fixed bug, and a negative label denotes that an online thread is useless for a fixed bug. EXPSOL implements SVM based on LIBSVM [7]. It is a popular library for SVMs.

Based on the training data, SVM constructs a hyperplane that separates the training data into useful pairs and useless pairs. The maximal margin hyperplane is as follow:

$$w = \sum_{i=1}^{n} \alpha_i l_i \phi(\vec{x}_i)) \quad (8)$$

In the above equation, α are real values, and ϕ is a mapping from the input space to a feature space. To locate the maximal margin hyperplane w, we need to maximize:

$$\sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j l_i l_j \langle \phi(\vec{x}_i), \phi(\vec{x}_j) \rangle \quad (9)$$

subject to

Algorithm 3 Candidate Set Filtering

Input:
A inputted bug report c
Exception tree nodes set N
Online resource set R
Output:
Candidate resource set R^*

```
Extract exceptions keywords set EP_c from c
Locate exception layer nodes N_EP ∈ N
while n_i ∈ N_EP do
    if n_i ∈ EP_c then
        Get tagged resources R_i ∈ R of n_i
        R* = R* + R_i
    end if
end while
return R*
```

$$\sum_{i=1}^{n} \alpha_i l_i = 0, \alpha_i > 0 \quad (10)$$

The decision function is expressed as follow:

$$f(x) = sign(\sum_{i=1}^{n} \alpha_i l_i \langle \phi(\vec{x}_i), \phi(\vec{x}) \rangle - b) \quad (11)$$

2) Candidate Set Filtering: Currently, there are more than 1 million exception-related forum threads and issues. As a result, it is rather time consuming and often unnecessary to compare those resource items one by one. To handle the problem, before the recommendation, we propose an algorithm to select only a subset of resource items. In particular, as shown in Algorithm 3, for each newly reported bug c, we link c with corresponding nodes of the exception tree, since most exception-related bug reports c and online resources r have exception name and keywords, and these keywords are also useful in identifying exception-related bugs. From the exception tree, we select the exception-layer nodes to match its exception name with keywords. As a result, only subtree leaf nodes of the linked nodes are selected as our recommendation candidates.

3) Results Ranking: Our trained model classifies online threads into useful ones and useless ones, and we rank the results in the following way. Zadrozny and Elkan [29] pointed that classifier scores can be transformed to estimate the probabilities of its classified types. For each pair of a newly reported bug and a thread, Equation (12) calculates a decision value, and such value reflects the confidence of our trained model on to what degree the thread is useful.

$$f_d(x) = \sum_{i=1}^{n} \alpha_i l_i \langle \phi(\vec{x}_i), \phi(\vec{x}) \rangle - b \quad (12)$$

Following the guideline of Zadrozny and Elkan [29], we define a sigmoid function to calculate the final ranks:

TABLE I
THE DATASET IN OUR EVALUATION

	GitHub Issues	StackOverflow Threads
Date Source	GHTorrent	Stackexchange
Datetime	Jun. 2015	Mar. 2015
Total Issues(Threads)	24,156,972	8,052,478
Exception Related	938,898	826,185
Selected Pairs	2,000	
Positive Pairs	1,200	
Negative Pairs	800	

$$f(d) = \frac{1}{1 + e^{-\beta(d-\gamma))}} \tag{13}$$

where β and γ are parameters, and d is calculated by Equation (12).

IV. EVALUATION

In this section, we first present our experimental settings and then analyze the experiment results. We totally conduct two evaluations to answer following questions:

- RQ1. How effective are EXPSOL's internal features?
- RQ2. How does EXPSOL improve existing approaches?

A. Dataset

We retrieve StackOverflow's threads from the Stackexchange archive[8] and retrieve the issues of GitHub from its mirror, GHTorrent[9] [14]. Table I shows the dataset in our evaluation. Row "Total Issues(Threads)" lists total issues and threads. Row "Exception Related" shows number of exception-related bugs and threads. In total, there are millions of exception-related issues and threads. As our underlying classification technique is supervised learning, it requires labeled data for training. The quality of the labeled training data is important, but it is often tedious and time-consuming to label data manually. After the investigation, we find that when some GitHub issues are closed, their discussions can contain related threads on StackOverflow, and vice versa. Furthermore, we find that if a closed GitHub issue mentions a StackOverflow thread and vice versa, the referred resource item is very likely to explain the resource item from the other side. As a result, we programmatically extract such pairs from exception-related items and apply carefully manually checking to filter out the positive pairs. In a positive pair, the forum thread should be the correct solution to the reported exception-related issue. For negative pairs, we are also able to extract exception context from one GitHub issue of the negative pair and treat StackOverflow thread in the pair as the wrong solution. As shown in Table I, in total, we prepare 2,000 pairs of GitHub issues and StackOverflow threads as our dataset. It consists of 1,200 positive pairs, and 800 negative pairs.

B. Performance Metrics

We use a list of performance metrics from the research areas of information retrieval and recommendation systems as follows:

[8]https://archive.org/details/stackexchange
[9]http://ghtorrent.org/

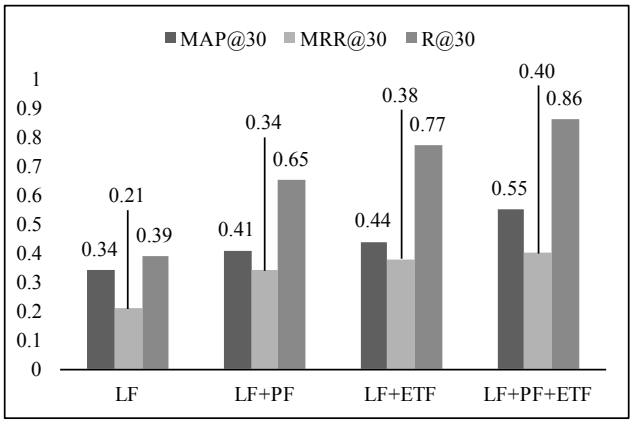


Fig. 4. The results of different models constructed by different feature(s)

1) Mean Average Precision (MAP): Average Precision is the average of precisions computed at the point of each of the relevant documents in the ranked sequence:

$$AP@N = \frac{\sum_{i=1}^{N}(P(i) \times rel(r))}{R} \tag{14}$$

Where r is the rank, N is the number of retrieved documents, R is the number of relevant documents, $rel()$ is a binary function on the relevance of a given rank, and $P(r)$ is precision at a given cut-off rank. The *Mean Average Precision* for N documents is the average of the *Average Precision* of each query:

$$MAP@N = \frac{1}{|Q|}\sum_{i=1}^{|Q|} AP_i@N \tag{15}$$

2) Mean Reciprocal Rank (MRR): The *Reciprocal Rank* of a query response is the multiplicative inverse of the rank of the first correct answer. The *Mean Reciprocal Rank* is a statistical measure that averages the *Reciprocal Rank* for each query in the query set Q:

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{16}$$

3) Recall (R): Recall denotes the fraction of the relevant results that are retrieved. In our experiments, we consider *recall* as the percentage of the exception-related bugs for which the solutions are recommended correctly.

C. RQ1. The Significance of Features

We use the labeled data generated in Table I as the ground truth to validate our approach and apply a five-fold cross-validation. Following the guideline of Golub *et al.* [12], we randomly partition the pairs into five equal size subsamples, so each subsample consists of 4,00 pairs. In each iteration, we select a subsample as the validation set, and the remaining pairs as the training set. The same training set is used for the various feature combinations in each phase of the cross-validation. In Section III-D, we define three types of features such as the lexical feature (LF), the program features (PF),

TABLE II
RESULTS OF EXPERIMENTS ON EXISTING APPROACHES

Approach	Metric	Top 10	Top 20	Top 30
Google	MAP	0.3821	0.3438	0.3228
	MRR	0.1836	0.1891	0.1921
	R	72.00%	72.00%	74.50%
StackOverflow	MAP	0.1738	0.1529	0.1399
	MRR	0.1083	0.1109	0.1134
	R	15.00%	22.00%	22.00%
Cordeiro et al.	MAP	0.4332	0.4288	0.3683
	MRR	0.2518	0.2572	0.2599
	R	25.50%	27.00%	28.00%
EXPSOL	MAP	**0.6043**	0.5643	0.5482
	MRR	0.3832	0.3895	**0.3954**
	R	78.50%	83.50%	**86.00%**

and the exception tree features (ETF). As shown in Figure 4, we totally trained four SVM models based on different combinations of features and measure the metrics at top 30.

Figure 4 shows the significance of our features. Our results show that the combination of LF, PF, and EFT achieves the best. When two types of features are considered, the results are better than only lexical feature. From the results, compared with total text similarity calculated in the lexical feature, program features perform better. This is because an exception context contains more program-related information which describes exception more accurate like code fragments or stack trace than natural language description. Also the combinations with EFT (*i.e.*, LF+ETF) is better than the combination without EFT (*i.e.*, LF+PF). It's easy to find that exception tree features are important and help a lot especially in improving the Mean Reciprocal Rank (MRR) and Recall (R). We consider this is mainly because the information structured in the exception tree is accurate for and good at describing and locating an exception. Programming languages, tags, and exceptions play a more important role in the recommendation than the general text description and programming code.

D. RQ2. The Improvement over Existing Approaches

In this section, we compare our approach with other state-of-the-art approaches such as the Google search engine, the search engine of StackOverflow, and a Lucene-based retrieval approach proposed by Cordeiro *et al.* [9]. From Table I, we select 2,00 GitHub issues as our experiment input dataset. We apply the approach of Cordeiro *et al.* to search the dataset in Table I, but cannot limit the search scope of the Google search engine and the search engine of StackOverflow.

To simulate the real development, we ask three experienced programmers to build queries for the selected fixed bugs. The programmers all have more than 5 years of development experience. For the Google search engine and the internal search engine of StackOverflow, the three programmers configure queries from exception messages with the pattern of "*ExceptionName* keywords". We limit the search scope of the Google search engine within StackOverflow with the `site` keyword. Cordeiro *et al.* recommend StackOverflow threads based on tokens that are taken from stack traces, code samples and texts. We limit its search scope with the dataset in Table I.

The results in Table II indicate that our approach achieves better results than other approaches. For the top 10 results, our approach achieves the highest Mean Average Precision (MAP) 0.6043, while the results of other approaches are all below 0.45. For the top 30 results, although the Mean Reciprocal Rank (MRR) and Recall (R) results of most the approaches are better than their results for the top 10 and top 20 results, their results still cannot catch up our approach. Cordeiro et al. have relatively stable Mean Reciprocal Rank (MRR) results but low Recall (R) values. It is interesting to find that, the internal search engine of StackOverflow performs much lower than Google, which highlights the performance of the Google search engine.

V. THREATS TO VALIDITY

The threat to internal validity includes the constructed queries in RQ2. While EXPSOL does not need a query, the compared approaches (*e.g.*, Google) do. As a result, we have to manually prepare their queries, which can be biased. To reduce the bias, we invited experienced programmers to prepare the queries. However, it shows our benefits, since it does not need programming experience to build queries.

The threat to external validity includes our selected subjects. Although we select thousands of online resources, we apply our approach only on limited online resources. The Internet-scale resources are much more than our selected subjects. To reduce the thread, we plan to integrate our approach with existing search engines, and conduct evaluations on more subjects in future work.

VI. RELATED WORK

Online forums host a rich information exchange, and there are existing different online threads retrieval approaches. Cong *et al.* [8] proposed a sequential patterns based classification method to detect questions in a forum thread. Yang *et al.* [28] took advantage of an adaptive feature-based matrix factorization framework to make thread recommendations. Bhatia *et al.* [3] proposed a model for online thread retrieval based on inference networks that utilized the structural properties of forum threads. Albaham *et al.* [1] adopted several voting techniques that had been applied in ranking aggregates tasks such as blog distillation and expert finding. Our approach recommends online threads to assist developers in fixing exception-related bugs, complementing the above approaches.

There are also some approaches that recommend software artifacts to assist various tasks. These approaches can be divided into retrieval filtering approaches and content-based approaches. For the retrieval filtering approaches, they mainly focus on constructing proper queries from context information of programming exceptions, searching with the queries and filtering out results [4]. Several existing search engines, such as Bing, Google or internal search engines of online communities, will accept the queries and return a group of results [20], [22]. With the help of ranking models and filtering strategies,

the highest scoring results are recommended [24]. Rahman *et al.* [23] used APIs from Google to recommend relevant context from web pages about programming errors and exceptions. SurfClipse [21] is a context-aware meta search engine with the support from Bing, Yahoo and Google APIs. Retrieval filtering approaches dependent on third party services so heavy that their performance is limited.

For the content-based approaches, they try to calculate the similarity between resources and programming errors in the IDE [15]. Seahawk [19] is an Eclipse plugin which supports an integrated and largely automated approach to assisting programmers in using StackOverflow. Monperrus *et al.* [18] imported StackOverflow documents from the public data dump and built document index with the help of Apache Lucene and TF-IDF. Cordeiro *et al.* [9] developed a tool that integrated recommendation of question/answer web resources in Eclipse, according to the context of their exception stack traces. Correa *et al.* [10] proposed an approach based on textual similarly analysis and contextual data analysis to StackOverflow question recommendation for an incoming programming bug. However, most of these approaches are constructed for IDE or other specific environments, they may not work for programming exception related issues or bug reports.

VII. Conclusion and Future Work

In this paper, we propose an approach, called EXPSOL, which recommends forum threads for exception-related bug reports. The two popular communities, StackOverflow and GitHub, are selected to evaluate our approach. Differing from the previous approaches, we construct an exception tree to link resources from the two communities. Exception tree based features are considered in our SVM training model to help improve the results. The evaluation results show the effectiveness of our approach. As for future work, we will try to add more details to our exception tree, such as versions of programming languages or external libraries. Moreover, it would be interesting to enrich our approach with resources from some other online communities or domains.

Acknowledgement

We thank the anonymous reviewers for their constructive comments. Beijun Shen is the corresponding author. This research is supported by 973 Program in China (Grant No. 2015CB352203) and National Natural Science Foundation of China (Grant No. 61472242). Hao Zhong is sponsored by National Nature Science Foundation of China No. 61572313 and the grant of Science and Technology Commission of Shanghai Municipality (No. 15DZ1100305).

References

[1] A. T. Albaham and N. Salim. Adapting voting techniques for online forum thread retrieval. In *Prod. 1st AMLTA*, pages 439–448, 2012.

[2] M. Allamanis and C. Sutton. Why, when, and what: analyzing stack overflow questions by topic, type, and code. In *Prod. 10th MSR*, pages 53–56, 2013.

[3] S. Bhatia and P. Mitra. Adopting inference networks for online thread retrieval. In *Prod. 24th AAAI*, 2010.

[4] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer. Example-centric programming: integrating web search into the development environment. In *Prod. 28th CHI*, pages 513–522, 2010.

[5] B. Cabral and P. Marques. Exception handling: A field study in Java and .NET. In *Prod. 21st ECOOP*, pages 151–175, 2007.

[6] E. C. Campos, L. B. L. D. Souza, M. A. Maia, and L. B. L. D. Souza. Nuggets miner: Assisting developers by harnessing the stackoverflow crowd knowledge and the github traceability. In *Proc. CBSoft - Tool Session*, 2014.

[7] C. Chang and C. Lin. LIBSVM: A library for support vector machines. *ACM TIST*, 2(3):27, 2011.

[8] G. Cong, L. Wang, C. Lin, Y. Song, and Y. Sun. Finding question-answer pairs from online forums. In *Prod. 31st SIGIR*, pages 467–474, 2008.

[9] J. Cordeiro, B. Antunes, and P. Gomes. Context-based recommendation to support problem solving in software development. In *Prod. 3rd RSSE*, pages 85–89, 2012.

[10] D. Correa and A. Sureka. Integrating issue tracking systems with community-based question and answering websites. In *Proc. 22nd ASWEC*, pages 88–96, 2013.

[11] M. Goldman and R. C. Miller. Codetrail: Connecting source code and web resources. *J. Vis. Lang. Comput.*, 20(4):223–235, 2009.

[12] G. H. Golub, M. Heath, and G. Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, 1979.

[13] C. Gómez, B. Cleary, and L. Singer. A study of innovation diffusion through link sharing on stack overflow. In *Proc. 10th MSR*, pages 81–84, 2013.

[14] G. Gousios. The ghtorrent dataset and tool suite. In *Prod. 10th MSR*, pages 233–236, 2013.

[15] B. Hartmann, D. MacDougall, J. Brandt, and S. R. Klemmer. What would other programmers do: suggesting solutions to error messages. In *Prod. 28th CHI*, pages 1019–1028, 2010.

[16] M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf. Support vector machines. *Intelligent Systems and their Applications*, 13(4):18–28, 1998.

[17] M. Mäntylä, J. Vanhanen, and C. Lassenius. Bad smells - humans as code critics. In *Prod. 20th ICSM*, pages 399–408, 2004.

[18] M. Monperrus, A. Maia, R. Rouvoy, and L. Seinturier. Debugging with the crowd: A debug recommendation system based on stackoverflow. *ERCIM News*, 2014(99), 2014.

[19] L. Ponzanelli, A. Bacchelli, and M. Lanza. Seahawk: Stack overflow in the ide. In *Prod. 35th ICSE*, pages 1295–1298, 2013.

[20] D. Poshyvanyk, M. Petrenko, and A. Marcus. Integrating cots search engines into eclipse: Google desktop case study. In *Prod. 2nd IWICSS*, page 6, 2007.

[21] M. M. Rahman and C. K. Roy. Surfclipse: Context-aware meta-search in the ide. In *Prod. 30th ICSME*, pages 617–620, 2014.

[22] M. M. Rahman and C. K. Roy. Recommending relevant sections from a webpage about programming errors and exceptions. In *Proc. 25th CASCON*, pages 181–190, 2015.

[23] M. M. Rahman, S. Yeasmin, and C. K. Roy. An ide-based context-aware meta search engine. In *Prod. 20th WCRE*, pages 467–471, 2013.

[24] M. M. Rahman, S. Yeasmin, and C. K. Roy. Towards a context-aware ide-based meta search engine for recommendation about programming errors and exceptions. In *Prod. CSMR-WCRE*, pages 194–203, 2014.

[25] M. S. Uddin, C. K. Roy, K. Schneider, A. Hindle, et al. On the effectiveness of simhash for detecting near-miss clones in large scale software systems. In *Prod. 18th WCRE*, pages 13–22, 2011.

[26] M. Umarji, S. E. Sim, and C. V. Lopes. Archetypal internet-scale source code searching. In *Prod. 20th OSS*, pages 257–263. 2008.

[27] B. Vasilescu, A. Serebrenik, P. T. Devanbu, and V. Filkov. How social q&a sites are changing knowledge sharing in open source software communities. In *Proc. CSCW*, pages 342–354, 2014.

[28] D. Yang, M. Piergallini, I. Howley, and C. Rose. Forum thread recommendation for massive open online courses. In *Prod. 7th ICEDM*. Citeseer, 2014.

[29] B. Zadrozny and C. Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Prod. 8th KDD*, pages 694–699, 2002.

[30] H. Zhong and Z. Su. An empirical study on real bug fixes. In *Proc. 37th ICSE*, pages 913–923, 2015.

[31] J. Zhu, B. Shen, X. Cai, and H. Wang. Building a large-scale software programming taxonomy from stackoverflow. In *Prod. 27th SEKE*, pages 391–396, 2015.