

# Leveraging Official Content and Social Context to Recommend Software Documentation

Jing Li, Zhenchang Xing and Ashad Kabir, *Member, IEEE*

**Abstract**—For an unfamiliar Application Programming Interface (API), software developers often access the official documentation to learn its usage, and post questions related to this API on social question and answering (Q&A) sites to seek solutions. The official software documentation often captures the information about functionality and parameters, but lacks detailed descriptions in different usage scenarios. On the contrary, the discussions about APIs on social Q&A sites provide enriching usages. Moreover, existing code search engines and information retrieval systems cannot effectively return relevant software documentation when the issued query does not contain code snippets or API-like terms. In this paper, we present CnCXL2R, a software documentation recommendation strategy incorporating the content of official documentation and the social context on Q&A into a learning-to-rank schema. In the proposed strategy, the content, local context and global context of documentation are considered to select candidate documents. Then four types of features are extracted to learn a ranking model. We conduct a large-scale automatic evaluation on Java documentation recommendation. The results show that CnCXL2R achieves state-of-the-art performance over the eight baseline models. We also compare the CnCXL2R with Google search. The results show that CnCXL2R can recommend more relevant software documentation, and can effectively capture the semantic between the high-level intent in developers' queries and the low-level implementation in software documentation.

**Index Terms**—Software documentation, recommendation systems, question and answering sites, ranking model

## 1 INTRODUCTION

With the emergence of Web 2.0 in modern software development, nowadays the behavior of developers is changed in relation to how they look for knowledge to fulfill their information needs [1], [2], as there are large number of accessible official software documentation (e.g., Application Programming Interface (API) documentation<sup>1</sup>, language tutorials<sup>2</sup>, and language specification<sup>3</sup>) and social media resources (e.g., question and answering sites, personal blogs and technique forums). The official resources provide important information about functionality, structure and parameters for APIs [3], [4]. The social resources provide enriching context in which the developers learn, preserve and share knowledge about software development and maintenance [5]–[7].

Official documentation is an important resource for developers to learn appropriate ways to use an unfamiliar API [8], [9]. Some studies had investigated API learning obstacles and found that there is a mismatch between the needs of consumers and the knowledge provided in software documentation [8], [10]. Robillard *et al.* [8] found that boilerplate member-level documentation will often not answer the query, and waste developer's time. Treude *et al.* [11] developed a technique to extract development tasks to navigate official

documentation because of the frustration of documentation structure, format and presentation. In addition, many recent studies [9], [12], [13] have targeted the recovery of traceability links between API and its learning resources. Most of existing works recover the traceability via a specific pair, such as <API, section fragment> pair [13], [14] and <API, keyword> pair [15]. These approaches often take advantage of the content and the context in software documentation but not the advantage of the context in social resources [10], [11]. On the contrary, recent work [15] only takes into account social resources but not the content of software documentation. Developers often issue queries using natural language [15], [16]. However, these approaches mainly accept API-like terms as input query and do not provide any support for natural language query [4], [17], [18]. Thus, the responses generated by them are very low quality.

Some studies [5], [6] show that the developers always benefit from social media such as online question and answering sites when they encountered programming problems. One of the most popular social media for this purpose is Stack Overflow<sup>4</sup>, which is an important question and answering venue for developers sharing knowledge on software development. Treude *et al.* [10] proposed a technique to extract insight sentences from Stack Overflow for augmenting API documentation, but do not support natural language query. Campos *et al.* [19] used the Stack Overflow data to recommend question-answer pairs as the solutions for API usage tasks instead of recommending software documentation. Although the Stack Overflow allows users to query in natural language, the users must need to manually check the tedious results to obtain the desired discussion

- J. Li is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798. E-mail: jli030@e.ntu.edu.sg
- Z. Xing is with the College of Engineering and Computer Science, Australian National University, Australia. E-mail: Zhenchang.Xing@anu.edu.au
- A. Kabir is with the School of Computing and Mathematics, Charles Sturt University, Australia. E-mail: akabir@csu.edu.au

1. <https://docs.oracle.com/javase/8/docs/api/index.html>

2. <https://docs.oracle.com/javase/tutorial/>

3. <https://docs.oracle.com/javase/specs/jls/se8/html/index.html>

4. <https://stackoverflow.com/>

```

Question: Right way to reinitialise transient variable
The Best Answer:
From the docs for java.io.Serializable:
Classes that require special handling during the serialization and deserialization process must implement
special methods with these exact signatures:
private void writeObject(java.io.ObjectOutputStream out)
    throws IOException
private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException;
private void readObjectNoData()
    throws ObjectStreamException;
The readObject method is responsible for reading from the stream and restoring the classes fields. It
may call in.defaultReadObject to invoke the default mechanism for restoring the object's non-static and
non-transient fields. The defaultReadObject method uses information in the stream to assign the fields
of the object saved in the stream with the correspondingly named fields in the current object. This
handles the case when the class has evolved to add new fields. The method does not need to concern
itself with the state belonging to its superclasses or subclasses. State is saved by writing the individual
fields to the ObjectOutputStream using the writeObject method or by using the methods for primitive
data types supported by DataOutput.
So basically, I think you want:
public class MySerializable implements Serializable{
    String value;
    transient String test = "default";
    public MySerializable() {
        test = "init";
    }
    private void readObject(java.io.ObjectInputStream in)
        throws IOException, ClassNotFoundException {
        in.defaultReadObject();
        test = "init";
    }
}
    
```

Fig. 1. A question and corresponding best answer from Stack Overflow.

threads. For example, the search engine returns 4225 pages with respect to the query “sort array in Java”<sup>5</sup>. Even worse, the software documentation related to the query is buried within the returned discussion threads.

When developers search software documentation in natural language, there is a mismatch between the high-level intent in developers’ queries and the low-level implementation in software documentation [8]. Moreover, a developer may want to complete a programming task with an unfamiliar API, and he may issue a query that does not contain any words found in the desired software documentation. The traditional information retrieval systems which consider only the textual similarity as the search criterion, cannot return the desired software documentation because the cosine similarity is equal to 0 in the *tf-idf* vector space. For example, consider the query “reinitialise transient variable”, there is no Java API documentation which contains all the three keywords of this query. Thus, we cannot obtain even a single relevant API documentation in Java documentation corpus by this query using traditional retrieval systems. Likewise, the state-of-the-art API usage miner [16] cannot return any relevant API sequence based on the code corpus of Github, but gives prompt of “Note: your query may not be supported by Java SDK library”<sup>6</sup>. However, a developer has implemented the task using the class `java.io.Serializable` and method `readObject` on Stack Overflow<sup>7</sup> as shown in Fig. 1. Therefore, considering the context of APIs on social platforms seems to be more appropriate in a software documentation recommendation strategy.

In this paper, we present CnCXL2R, a recommendation strategy that incorporates the content of software documentation and social context on Stack Overflow into a learning-to-rank schema. Given a natural language query, CnCXL2R recommends software documentation through two key steps: candidate software documentation selection and recommendation by learning to rank. *First*, to bridge the lexical gap between query and documentation, CnCXL2R generates candidate software documents based on not only official content but also social context. *Second*, 22 features are extracted to measure the relevance of a software document

to a query. In particular, to further bridge the lexical gap, we utilize a neural language model [16], [20] to embed the context of software documentation on Stack Overflow to capture the semantical similarity between a query and a candidate software document. *Finally*, a ranking model is trained by a learning-to-rank manner to recommend top-*k* software documents. To evaluate the performance of CnCXL2R, we conducted a large-scale automatic evaluation using the real discussions on Stack Overflow. We also conducted a user study to compare the recommendation performance with Google search. In short, we make the following contributions:

- We propose CnCXL2R, a novel software recommendation strategy incorporating the official content of documentation and social context on Stack Overflow into a learning-to-rank schema. It can perform well with natural language queries comparing with traditional code search.
- We investigate four types of features, *i.e.*, statistical-based, textual, context and popularity features, for learning the software documentation ranker. All the four types of features are easy to derive and enabling real-time recommendation.
- We conduct a large-scale automatic evaluation to evaluate the performance of CnCXL2R and investigate the effects of different feature groups and embedding parameters. CnCXL2R achieves the state-of-the-art results against 8 baseline methods.
- We conduct a user study to compare the recommended results by CnCXL2R with Google search. Results show that CnCXL2R significantly outperforms Google search in software documentation retrieval task.

The remainder of this paper is organized as follows. In Section 2, we describe the details of CnCXL2R. In Section 3, we explain the design of experiments. In Section 4, we present the results. Section 5 discusses the threats to validity. After summarizing the related work in Section 6, we conclude the paper in Section 7.

## 2 METHODOLOGY

In this section, we present our approach in details. We first introduce the overall architecture. Then we describe the approach to selecting candidate software documentation. Finally, we present our learning-to-rank approach for software documentation recommendation.

### 2.1 Overview

Fig. 2 shows the architecture of the proposed approach, which consists of two core phases: candidate software documentation selection and recommendation by learning-to-rank model.

Our idea is based on the consideration that social context can provide complementary information for official software documentation. In the phase of candidate documentation selection, we take into account three factors: the content of a software document, local and global context of a software document. This process effectively recommends possible software documents for the programming task in a query.

5. <https://stackoverflow.com/search?q=sort+array+in+java>

6. <http://211.249.63.55/>

7. <http://stackoverflow.com/questions/18893032>

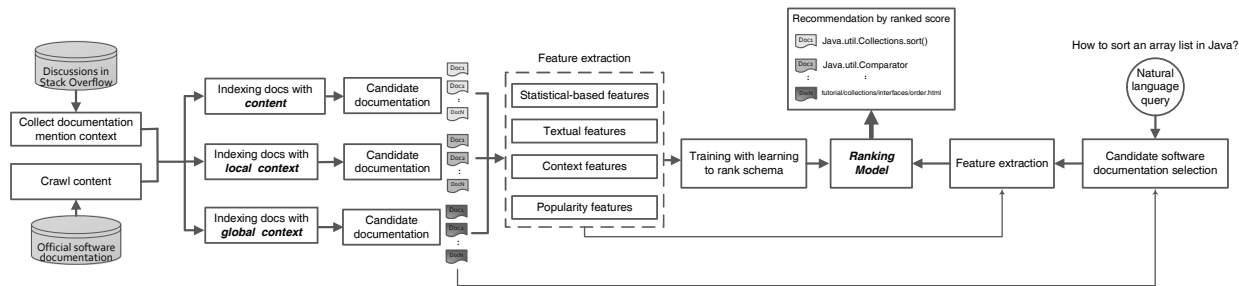


Fig. 2. The architecture of CnCXL2R.

153 In the phase of recommendation, we train a ranking  
 154 model using four types of features, *i.e.*, statistical-based,  
 155 textual, context and popularity features. The selected candi-  
 156 date software documents in the first phase are ranked using  
 157 the trained model. The input of our approach is a natural  
 158 language query about programming task and the output is  
 159 a ranked list of official software documentation. Training  
 160 the ranking model is an off-line process, while ranking the  
 161 candidate software documentation is a run-time process that  
 162 starts when a query is issued by a developer.

164 **2.2 Candidate Software Documentation Selection**

165 Given a query, candidate software documentation selec-  
 166 tion is a process of selecting a subset of documents related  
 167 to the programming task in the query. In this paper, soft-  
 168 ware documentation refers to API documentation, language  
 169 tutorial and language specification. API documentation con-  
 170 tains all the information about the functions, classes, return  
 171 types, arguments and so on. Language tutorials are practical  
 172 guides for programmers who want to use a programming  
 173 language to create applications. Language tutorials include  
 174 hundreds of complete, working examples, and dozens of  
 175 lessons. Language specification is the definitive technical  
 176 reference including the semantics of all types, statements,  
 177 and expressions, as well as threads and binary compatibility.  
 178 We consider the following three approaches to select the  
 179 candidate software documents.

180 **2.2.1 Selection by Content**

181 For a query, the content of documentation reflects the sur-  
 182 face relevance between the query and software documenta-  
 183 tion. To retrieve software documents for a query, we need to  
 184 crawl the content of a corpus of software documentation. We  
 185 crawl all documents in our corpus (detailed in Section 3.1)  
 186 and perform following pre-processing.

- 187 • Usually, there are some code fragments in software  
 188 documentation. These code fragments commonly in-  
 189 clude class names and method names. We retain code  
 190 fragments in content because it can provide hints for  
 191 a query.
- 192 • The stopwords are general words and no meaning  
 193 if they are used alone and appear frequently in text.  
 194 Thus, we remove the English stopwords provided by  
 195 the Natural Language Toolkit [21], which contains  
 196 127 words (*e.g.*, *all*, *just*, *being*,...).  
 197 • Stemming can potentially increase the discriminative  
 198 power of root words and reduce inflected words to  
 199 their word stem. We reduce a word to its root using  
 200 Porter Stemming Algorithm [22].

201 Given a query, we use Lucene<sup>8</sup> engine and Latent Di-  
 202 richlet allocation (LDA) [23] model to retrieve candidate  
 203 software documentation.

204 *Lucene engine:* We index each software documentation  
 205 as an index document in Apache Lucene engine. For each  
 206 query, we retrieve the top 10 results from corpus using  
 207 Lucene with BM25 scoring function.

208 *LDA model:* We use LDA model to represent the query  
 209 and the content of software documentation as vectors in  
 210 topic space. Then we retrieve the top 10 results based on  
 211 cosine similarity of topic distributions.

212 **2.2.2 Selection by Local Context**

213 Software documentation is mainly written to effectively  
 214 capture the information about functionality, structure and  
 215 parameters, but lacking insights about usage scenarios and  
 216 cautions [10]. Stack Overflow is a popular question and an-  
 217 swering site where developers ask programming questions to  
 218 seek solutions. The content of discussions on Stack Overflow  
 219 provides enriching context to mine usage scenarios and  
 220 cautions of APIs [9].

221 **DEFINITION 1 (Discussion Thread).** A discussion thread  
 222 consists of a question and all its answers. The question  
 223 and answers are referred to as posts.

224 When a software document appears in a discussion thread,  
 225 its surround texts provide enriching context for its usage  
 226 scenarios. Now we give the definition of concept of local  
 227 context for a software document.

228 **DEFINITION 2 (Local Context).** If a software document is  
 229 mentioned in a best answer, the texts of the question  
 230 (title and body) and the best answer are regarded as the  
 231 local context of the software document.

232 The definition of local context is based on the consid-  
 233 eration that the quality of best answer is better than other  
 234 answers in the discussion thread. The text of the best answer  
 235 is the immediate context when a software document appears  
 236 in a best answer. On the other hand, the question title  
 237 and question body, describing the programming problem  
 238 in detail, provide cues to reflect the relevance between the  
 239 problem and the software documentation in the best answer.

240 For example, the best answer<sup>9</sup> on Stack Overflow, menti-  
 241 ons two API documents: `java.url.regex.Pattern` and  
 242 `java.util.ArrayList`. This discussion thread consists of  
 243 a question and three associated answers. In terms of the  
 244 definition, we only consider question title, question body

8. <http://lucene.apache.org/>  
 9. <http://stackoverflow.com/questions/18625462>

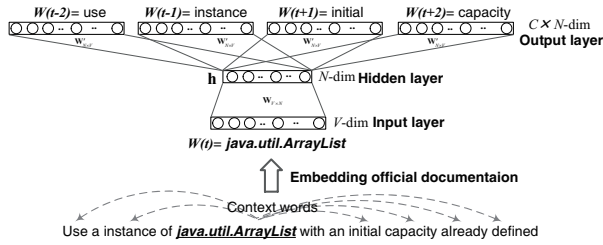


Fig. 3. Training word embedding example with the skip-gram model.

245 and the body of the best answer as local context for the two  
246 API documents, excluding the other two answers.

247 We collect all local contexts for the software documents  
248 in our corpus. Each local context of a software document  
249 is considered as an independent document to be retrieved.  
250 Given a query, we use Lucene engine and LDA model to  
251 retrieve the most relevant local context and pick out the  
252 software documents in the local context as recommendation.

253 *Lucene engine:* We index every local context as an index  
254 document using Apache Lucene engine. For each query, we  
255 retrieve the top 10 software documents from the returned  
256 local contexts using Lucene with BM25 scoring function.

257 *LDA model:* Query and local context are represented  
258 as vectors in topic space. We retrieve the top 10 software  
259 documents from the returned local contexts based on cosine  
260 similarity of topic distributions.

### 261 2.2.3 Selection by Global Context

262 A specific software document may be mentioned in mul-  
263 tiple discussion threads. For example, the documentation  
264 `java.util.ArrayList` was mentioned 906 times. Thus,  
265 there are many local contexts for a specific software docu-  
266 ment.

267 **DEFINITION 3 (Global Context).** The global context of a  
268 software document is the collection of all its local con-  
269 texts.

270 For the global context, we use a neural language model  
271 for learning term and documentation representations. The  
272 key idea is based on Harris' distributional hypothesis [24],  
273 which states that words in the same context tends to have  
274 similar meanings and similar words have similar vector  
275 representations. Traditionally, language models represent  
276 each term as a feature vector using one-hot representation,  
277 where a vector element that corresponds to the observed  
278 word is equal to 1 and 0 otherwise [25]. Recently, neu-  
279 ral language models have been proposed to address low-  
280 dimensional, distributed embedding of words [26], [27].  
281 These approaches take advantage of neural language mod-  
282 els to capture both syntactic and semantic relationships  
283 between words. Mikolov's continuous bag-of-words and  
284 skip-gram language models [28], [29] are powerful and  
285 efficient approaches to learn distributed word embeddings.

286 Fig. 3 illustrates the training procedure with the  
287 skip-gram model when it reaches the current word  
288 `java.util.ArrayList`<sup>10</sup>. We use  $\mathbf{V}_{w_t}$  to denote the input  
289 of vector of the only word  $w_t$  on the input layer.  $N$  is the  
290 hidden layer size.  $V$  is the vocabulary size.  $C$  is the number

10. <http://stackoverflow.com/questions/18625462/get-a-particular-list-of-string-using-regex-in-java/18625626#18625626>

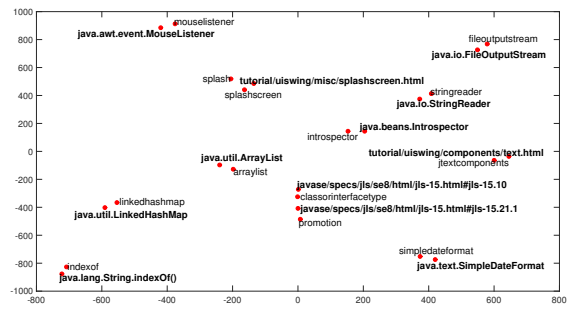


Fig. 4. A 2D projection of embedding natural language words and software documents using PCA.

of words in the context. The output of hidden layer can be  
written as

$$291 \quad \mathbf{h} = \mathbf{W}^T \mathbf{x} = \mathbf{V}_{w_t}^T \quad (1) \quad 292 \quad 293$$

294 where  $\mathbf{W}$  is a  $V \times N$  input  $\rightarrow$  hidden weight matrix.  $\mathbf{V}_{w_t}$   
295 is the vector representation of the input word  $w_t$ .

296 On the output layer, each output is computed using the  
297 hidden  $\rightarrow$  output matrix:

$$298 \quad p(w_{c,j} = w_{O,c} | w_t) = \frac{\exp(u_{c,j})}{\sum_{j'=1}^V \exp(u_{j'})} \quad (2) \quad 299$$

300 where  $w_t$  is the input word;  $w_{c,j}$  is the  $j$ -th word on the  $c$ -th  
301 panel of the output layer;  $w_{O,c}$  is the actual  $c$ -th word in the  
302 output context word.  $u_{c,j}$  is the net input of the  $j$ -th unit on  
the  $c$ -th panel of the output layer,

$$303 \quad u_{c,j} = \mathbf{V}_{w_j}^T \cdot \mathbf{h}, \text{ for } c = 1, 2, \dots, C \quad (3)$$

304 where  $\mathbf{V}_{w_j}^T$  is the output vector of the  $j$ -th word in the  
305 vocabulary,  $w_j$  and  $\mathbf{V}_{w_j}^T$  is taken from a column of the  
306 hidden  $\rightarrow$  output weight matrix,  $\mathbf{W}'$ .

307 When training the skip-gram model to predict  $C$  context  
308 words, the loss function is written as

$$309 \quad \begin{aligned} E &= -\log p(w_{O,1}, w_{O,2}, \dots, w_{O,C} | w_t) \\ &= -\log \prod_{c=1}^C \frac{\exp(u_{c,j_c^*})}{\sum_{j'=1}^V \exp(u_{j'})} \\ &= -\sum_{c=1}^C u_{j_c^*} + C \cdot \log \sum_{j'=1}^V \exp(u_{j'}) \end{aligned} \quad (4)$$

310 where  $j_c^*$  is the index of the actual  $c$ -th output context word  
311 in the vocabulary.

312 We use the skip-gram to learn embeddings of natural  
313 language words and software documentation. Fig. 4 illus-  
314 trates a 2-D projection of vectors of natural language words  
315 and software documentation in our dataset using principal  
316 component analysis (PCA). In the embedding space, sema-  
317 ntically close words are likewise close in the embedding  
318 space while they are not close based on surface simila-  
319 rity such as term overlapping and TF-IDF weight. Especi-  
320 ally, the vectors of the word and software documentation  
321 with same intent have the shortest distance. For example,  
322 the word "arraylist" is close to the API documentation  
323 `java.util.ArrayList`.

324 Following [30], we use bag-of-words model to average  
325 out the vectors of the individual words in a query. Given a  
326 query, we retrieve top 10 software documents based on the

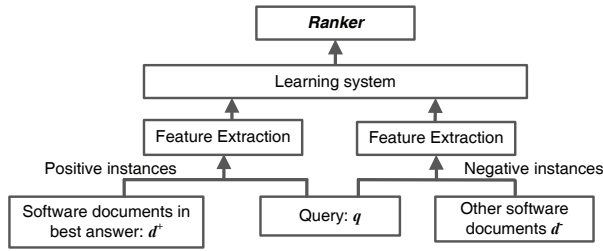


Fig. 5. Learning-to-rank architecture.

327 cosine similarity between the average vector and software  
328 documentation vector.  
329

### 330 2.3 Recommendation by Learning to Rank

331 Given a programming query, our task is to recommend a  
332 ranked list of software documents that are most relevant  
333 to the intent in the query. More specifically, we formul-  
334 ate the software documentation recommendation task as  
335 a learning-to-rank problem.

#### 336 2.3.1 Problem Formulation

337 For the ranking task, a successful approach is to treat it as a  
338 supervised machine learning problem [31]. The most typical  
339 setup is the supervised learning-to-rank schema, which is  
340 as follows: Assume that there is a corpus of documents. In  
341 the training process, a set of queries and retrieved lists are  
342 provided. Each query is associated with a set of retrieved  
343 documents with relevance judgments. In the learning-to-  
344 rank schema, each query-document pair is represented by  
345 a set of features. The learning-to-rank schema automatically  
346 learns the optimal way of combining these features. A set of  
347 such pairs is used to train a machine learning algorithm,  
348 then a ranking function is built to rank the documents  
349 pertaining to the query.

350 Fig. 5 shows the architecture of learning-to-rank process.  
351  $D = \{d_1, d_2, \dots, d_n\}$  represents the retrieved candidate soft-  
352 ware documents for a query  $q$ . The candidate set comes with  
353 their relevancy judgments, where the software documents in  
354 the best answer are annotated as positive instances ( $\langle q, d_+ \rangle$ )  
355 and negative for otherwise ( $\langle q, d_- \rangle$ ). Our goal is to build  
356 a ranking model which facilitates optimal ranking of the  
357 candidate list  $D$  for a query  $q$ . More formally, the task is to  
358 learn a scoring function  $F(q, d)$ :

$$359 F(q, d) = \sum_{k=1}^K \omega_k \cdot \phi_k(q, d) \quad (5)$$

360 where each feature  $\phi_k(q, d)$  measures a specific relationship  
361 between query  $q$  and candidate software documentation  $d$ .  
362  $\omega_k$  is the weight of the  $k$ -th feature (total  $K$  features), and  
363 is learned during the training. The optimization procedure  
364 of learning-to-rank tries to find the scoring function that  
365 can rank the relevant software documents at the top of  
366 candidate list.

367 There are mainly three approaches to learn scoring  
368 function, namely, pointwise [32]–[34], pairwise [35]–[37] and  
369 listwise [38]–[40]. Pointwise approach is the most simple  
370 way to build ranking model, which defines the loss function  
371 based on individual documentation. The pointwise com-  
372 monly be viewed as regression approach by minimizing

a loss function. Pairwise approach transforms the ranking  
373 problem to pairwise classification, taking candidate docu-  
374 mentation pairs as instances in learning instead of indivi-  
375 dual documentation. Listwise approach generates a candi-  
376 date list through the comparison between two documents.  
377 Listwise approach takes lists as instances in learning and  
378 loss function is defined on basis. Our approach falls within  
379 the category of pairwise.  
380

#### 381 2.3.2 Feature Extraction

382 In this section, we describe the details of feature engineering  
383 for training the ranking schema in our approach. Totally, we  
384 extract 22 features, which fall into four groups: Statistical-  
385 based, Textual, Context, and Popularity features as listed  
386 in Table 1. Furthermore, we divide the 22 features into 3  
387 categories: Q-D means that the feature is dependent on  
388 both query and candidate document, Q represents that the  
389 feature is calculated by query regardless of candidate docu-  
390 ment, D means that the feature only depends on candidate  
391 document regardless of query.

**Statistical-based Features.** We use 10 statistical-based fea-  
392 tures [20], [41] in our learning-to-rank schema, which are  
393 widely used in information retrieval community and shown  
394 in Table 1.

We define that  $q$  represents a query, which consists of  $t$   
396 terms  $q_1, q_2, \dots, q_t$ . The number of occurrences of the query  
397 term  $q_i$  in document  $d$  is denoted as  $c(q_i, d)$ . Document fre-  
398 quency  $df(q_i)$  reflects the number of documents containing  
399  $q_i$  in the document collection.  $|C|$  is the total number of  
400 documents in the document collection and  $|d|$  is the length  
401 (i.e., the number of terms) of document  $d$ . We obtain 10  
402 features (F1-F10) based on the above definitions. Note that  
403 F5, F6 and F7 belong to Q category, and the other 7 features  
404 belong to Q-D category.  
405

**Textual Features.** Textual features are the basic features  
406 used to judge relevancy between a query and a software  
407 document.

*textualSim:* This feature computes the textual similarity  
408 between a query and a software document using cosine  
409 similarity.  
410

$$411 \text{textualSim}(V_q, V_d) = \cos(V_q, V_d) = \frac{V_q^T V_d}{\|V_q\| \|V_d\|} \quad (6)$$

412 where  $V_q$  is a query vector and  $V_d$  is a vector of a software  
413 document based on bag-of-words model.  
414

*textualBM25:* BM25 [42] is a ranking function used by  
415 search engines to rank matching documents according to  
416 their relevance to a given search query. The BM25 score  
417 between query  $q$  and a software document  $d$  is computed  
418 as follows,  
419

$$420 \text{BM25}(q, d) = \sum_{q_i \in q} \frac{idf(q_i) \cdot c(q_i, d) \cdot (k_1 + 1)}{c(q_i, d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{avgdl}\right)} \cdot \frac{(k_3 + 1) \cdot c(q_i, d)}{k_3 + c(q_i, d)} \quad (7)$$

421 where  $avgdl$  is the average software documentation length  
422 in the entire document corpus.  $k_1, k_3$  and  $b$  are free param-  
423 eters. Following the benchmark of learning-to-rank system  
424 [41], we set  $k_1 = 2.5, k_3 = 0$  and  $b = 0.8$ .

TABLE 1  
Learning features.

Group	Feature name	Description	Category
Statistical-based	F1	$\sum_{q_i \in q \cap d} c(q_i, d)$	Q-D
	F2	$\sum_{q_i \in q \cap d} \log(c(q_i, d) + 1)$	Q-D
	F3	$\sum_{q_i \in q \cap d} \frac{c(q_i, d)}{ d }$	Q-D
	F4	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i, d)}{ d } + 1\right)$	Q-D
	F5	$\sum_{q_i \in q \cap d} \log\left(\frac{ C }{df(q_i)}\right)$	Q
	F6	$\sum_{q_i \in q \cap d} \log\left(\log\left(\frac{ C }{df(q_i)}\right)\right)$	Q
	F7	$\sum_{q_i \in q \cap d} \log\left(\frac{ C }{c(q_i, C)} + 1\right)$	Q
	F8	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i, C)}{ d } \cdot \log\left(\frac{ C }{df(q_i)}\right) + 1\right)$	Q-D
	F9	$\sum_{q_i \in q \cap d} c(q_i, d) \cdot \log\left(\frac{ C }{df(q_i)}\right)$	Q-D
	F10	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i, d)}{ d } \cdot \frac{ C }{c(q_i, C)} + 1\right)$	Q-D
Textual	<i>textualSim</i>	The textual similarity between a query and a software document based on bag-of-words model.	Q-D
	<i>BM25</i>	The BM25 score between a query and a software document.	Q-D
	<i>isClueInQuery</i>	Whether the clue word in the URL of a software document is contained in a query.	Q-D
	<i>isHashmark</i>	Whether the URL of a software document contains a fragment identifier (hashmark #).	D
	<i>docCategory</i>	One of categories (official APIs, official tutorials and official specifications).	D
	<i>numSlash</i>	Number of slashes in the URL of a software document.	D
	<i>lengthUrl</i>	Length of strings in the URL of a software document.	D
	<i>lengthDoc</i>	Number of terms in a software document.	D
Context	<i>lengthQuery</i>	Number of terms in a query.	Q
	<i>contextSim</i>	The context cosine similarity between a query and a software document.	Q-D
Popularity	<i>refFrequency</i>	The number of times that a software document is referenced on Stack Overflow.	D
	<i>isBest</i>	It is true when a software document ever occurred in best answers.	D

425 *isClueInQuery*: Whether the clue word in the URL of a  
426 software document is contained in the query. For exam-  
427 ple, *isClueInQuery* is true when the clue word “HashSet”  
428 in the URL [https://docs.oracle.com/javase/7/docs/api/java/util/](https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html)  
429 [HashSet.html](https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html) is contained in the query “Understanding  
430 *HashSet*”.

431 *isHashmark*: Whether the URL of a software document  
432 contains a fragment identifier (hashmark #). For example,  
433 *isHashmark* is true because the URL [https://docs.oracle.com/](https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html#getResourceAsStream-java.lang.String-)  
434 [javase/8/docs/api/java/lang/Class.html#getResourceAsStream-](https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html#getResourceAsStream-java.lang.String-)  
435 [java.lang.String-](https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html#getResourceAsStream-java.lang.String-) contains a hashmark #.

436 *docCategory*: The official software documentation con-  
437 sists of official APIs, official language tutorials and official  
438 language specifications. *docCategory* is equal to 1 when the  
439 documentation is an official API, 2 for official tutorial, and  
440 3 for official specification.

441 *numSlash*: Number of slashes in the URL of a software  
442 document.

443 *lengthUrl*: Length of strings in the URL of a software  
444 document.

445 *lengthDoc*: Number of terms in a software document.

446 *lengthQuery*: Number of terms in a query.

447 **Context Features.** After training the word2vec corpus, we  
448 can get the vector representation of each word. Suppose  
449 that the length (*i.e.*, the number of terms) of a query and  
450 a software document are  $N$  and  $M$ , respectively. In order to  
451 get the text vector, we average the word vectors in the text.  
452  $w_q$  and  $w_d$  denote the text vector of a query and a software  
453 document, respectively. We can get the context similarity be-  
454 tween the query and the software documentation as follows:

$$contextSim(w_q, w_d) = \cos(w_q, w_d) = \frac{\frac{1}{NM} \cdot \sum w_i \cdot \sum w_j}{\left\| \frac{\sum w_i}{N} \right\| \left\| \frac{\sum w_j}{M} \right\|} \quad (8)$$

455 **Popularity Features.** Much meta-data is available on Stack  
456 Overflow. We can extract features from these meta-data to  
457 measure the popularity of a software document.

458 *refFrequency*: This feature measures the number of times  
459 that a software document is referenced on Stack Overflow.

460 *isBest*: It is true when a software document was occurred  
461 in a best answer.

### 2.3.3 Learning a Ranker

462 **Learning a Ranker** 463  
464 In our approach, we train the ranking model using one  
465 of the state-of-the-art learning-to-rank algorithms, namely,  
466 LambdaMART [43], a boosted tree version of Lambda-  
467 Rank [44], that won the Yahoo! Learning to Rank Challenge.

468 In our approach, the input of learning-to-rank algo-  
469 rithm is the training instances which represent each query-  
470 documentation pair as a feature vector.  $d_i$  and  $d_j$  denote can-  
471 didate software documentation  $i$  and  $j$  in the given query  
472  $q$ , respectively. Based on feature extraction in Section 2.3.2,  
473  $x_i$  and  $x_j$  are feature vector for  $\langle q, d_i \rangle$  and  $\langle q, d_j \rangle$ , re-  
474 spectively. LambdaMART builds a regression tree to model  
475 the functional gradient of the cost function of interest, and  
476 evaluate at all the training pairs [43], [45]. The  $\lambda$ -gradients  
477 can be written as follows:

$$\lambda_{ij} = S_{ij} \left| \Delta NDCG \frac{\partial C_{ij}}{\partial \sigma_{ij}} \right| \quad (9)$$

478 where  $S_{ij} \in \{-1, 1\}$  is equal to 1 if the documentation  
479  $d_i$  is more relevant than the documentation  $d_j$ , and is  
480 equal to -1 if the documentaion  $d_i$  is less relevant than  
481 the documentation  $d_j$ .  $\Delta NDCG$  is the NDCG (Normalized  
482 Discounted Cumulative Gain) gained by swapping those  
483 two software documents. Let  $F(x)$  is the ranking function,  
484 and  $\sigma_{ij} = F(x_i) - F(x_j)$  is the difference in ranking scores  
485 for the pair of software documents.  $C_{ij} = F(x_j) - F(x_i) +$   
486  $\log(1 + e^{F(x_i) - F(x_j)})$  is the cross-entropy cost applied to  
487

488 the logistic of the difference of the scores. Each point then  
489 sums its  $\lambda$ -gradients for all pairs  $P$  in which it occurs:

490 
$$\lambda_i = \sum_{j \in P} \lambda_{ij} \quad (10)$$

491 A positive lambda indicates a push toward the top rank  
492 position and a negative lambda indicates a push toward the  
493 lower rank positions. Finally, the ranking function  $F(x)$  can  
494 be learned based on the  $\lambda$ -gradients and Newton-Raphson  
495 line step [43].

### 496 3 EXPERIMENTAL SETUP

497 In this study, we performed a set of experiments with the  
498 data from official software documentation websites and  
499 Stack Overflow website. We now describe our experimental  
500 setup: data collection, performance measures and compari-  
501 son baselines.

#### 503 3.1 Data Collection

504 There are two types of text data used in this research: the  
505 text of software documentation and the text of discussion  
506 threads on Stack Overflow.

507 **Corpus for Word Embedding.** In this research, we focus  
508 on Java software documentation on its official sites and  
509 Java discussion threads on Stack Overflow. The discus-  
510 sion threads on Stack Overflow provide enriching insights  
511 for software documentation by means of hyperlinks. The  
512 official documentation mainly provides information about  
513 functionality, structure and parameters instead of insights  
514 and feedbacks on Stack Overflow. Usually, a query is expres-  
515 sed in natural language, and much closer to the discussions  
516 on Stack Overflow. Thus, we use the discussion threads on  
517 Stack Overflow as the corpus for word embedding.

518 For the questions and answers on Stack Overflow, their  
519 quality significantly varies from post to post. In order to  
520 guarantee the quality of text, we extract these discussion  
521 threads using the datadump archive available on the Stack  
522 Exchange website<sup>11</sup> that satisfy the following criteria:

- 523 • The question tags contain "Java". This condition  
524 filters out the local and global context about "Java"  
525 from the sea of information.
- 526 • The score of question is larger than 0. This condition  
527 guarantees that at least a developer has voted the  
528 question as an "useful" question.
- 529 • The question has an answer which is accepted as the  
530 best answer. Meanwhile, the score of the best answer  
531 is larger than 0. This condition guarantees the quality  
532 of the best answer.
- 533 • The best answer must contain at least one hyperlink  
534 which links to official Java API documentation, or  
535 official Java tutorial, or official Java specification.

536 Based on the above criteria, we totally collect 30,272  
537 discussion threads based on data dump released on Au-  
538 gust 2015. We randomly select 24,217 discussion threads  
539 (account for 0.8) as training discussion threads and other  
540 6055 threads (account for 0.2) as test discussion threads. For

the training discussion threads, we take the following text  
pre-processing:

- 541 • Select question title, question body and best answer  
542 body as text document for embedding. 543
- 544 • Remove code snippets and english stop words, and  
545 change all characters to lowercase. 546
- 547 • Add the actual URL address after the anchor text of  
548 hyperlink of software documentation. 549
- 550 • Normalize the URLs of software documentation of  
551 different Java versions. For example, in our voca-  
552 bulary, we use "api\_docs/api/java/util/ArrayList.html"  
553 to label the three URLs ([https://docs.oracle.com/  
554 javase/7/docs/api/java/util/ArrayList.html](https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html), [https://docs.  
555 oracle.com/javase/8/docs/api/java/util/ArrayList.html](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html),  
556 and [https://docs.oracle.com/javase/6/docs/api/java/util/  
557 ArrayList.html](https://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html)).

558 We use these 24,217 discussion threads as training cor-  
559 pus for word embedding. To learn word representations,  
560 we use the skip-gram model implemented in Gensim<sup>12</sup>.  
561 The context window size is set to 10 and the minimal  
562 word frequency is 5. Finally, we collect 1,520 Java software  
563 documents from training discussion threads.

564 **Crawl Official Documentation.** After training the word  
565 embeddings, we get 1,520 Java software documents and we  
566 consider these software documents as our research corpus.  
567 We crawl these 1,520 software documents via the hyper-  
568 links in discussion threads. We take the following text pre-  
569 processing:

- 569 • Usually, a class or interface level of documentation  
570 is very long because it contains thorough method  
571 descriptions. For example, `java.util.ArrayList`  
572 ([https://docs.oracle.com/javase/8/docs/api/java/util/  
573 ArrayList.html](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html)) contains 31 methods in its html file.  
574 In such cases, we only crawl the class or interface  
575 description in the beginning of html file as the  
576 content of the URL.
- 577 • If an URL of a software document contains a  
578 fragment identifier (hashmark #), we crawl the  
579 content of the section that the hashmark identi-  
580 fies. For example, the URL of [https://docs.oracle.com/  
581 javase/8/docs/api/java/util/ArrayList.html#get-int-](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#get-int-)  
582 mentions the method `get(int index)`. We only crawl  
583 the section of `get(int index)` in the html file.

584 **Dataset for Learning A Ranker.** In this study, we only focus  
585 on the corpus consisting of 1,520 Java software documents  
586 as the demonstration of effectiveness of CnCxL2R. We leave  
587 the cold-start problem in the future research. Thus, we select  
588 the discussion threads whose best answers contain at least  
589 one software document of our corpus to learn a ranker.  
590 Based on the above setting, we collect 12,020 discussion  
591 threads for training a learning-to-rank model. In the corpus  
592 of test discussion threads, we collect 2,924 discussion thre-  
593 ads for testing our recommendations.

594 For a selected discussion thread, we assume that the  
595 question title contains sufficient information for the problem  
596 in the question. We consider the question title as the input  
597 query of CnCxL2R. The pair  $\langle query, software\ documentation \rangle$

11. <https://archive.org/details/stackexchange>

12. <https://radimrehurek.com/gensim/>

598 *in the best answer*> is treated as a positive instance and an  
 599 randomly sampled pair as a negative instance.  
 600

### 601 3.2 Performance Measures

602 We choose four performance metrics which are widely  
 603 used for recommendation evaluation: precision at position  
 604  $k$ , recall at position  $k$ , mean average precision (MAP) and  
 605 mean reciprocal rank (MRR) [46], [47].

606 **Precision at position  $k$  ( $P@k$ ):**  $P@k$  is a metric to measure  
 607 top  $k$  positions of a ranked list using two levels of relevance  
 608 judgment (relevant and irrelevant). For a natural language  
 609 query  $q$ , let  $D_k$  be the set of top- $k$  recommended official  
 610 documents and  $D_g$  be the set of ground-truth documents.  
 611  $P@k$  is calculated by  $P@k = \frac{|D_k \cap D_g|}{k}$ .

612 **Recall at position  $k$  ( $R@k$ ):**  $R@k$  is a measure for evaluating  
 613 the fraction of top- $k$  documents that are relevant to the  
 614 query that are successfully retrieved.  $R@k$  can be defined  
 615 as follows:  $R@k = \frac{|D_k \cap D_g|}{|D_g|}$ .

616 **Mean average precision (MAP):** MAP is defined as the  
 617 mean of average precision (AP) over a set of queries. Let  $Q$   
 618 be the set of all test queries. The set of relevant documents  
 619 for  $q_j \in Q$  is  $\{d_1, \dots, d_k, \dots, d_{m_j}\}$ , and  $R_{jk}$  is the set of  
 620 ranked results from the top results until you get to docu-  
 621 ment  $d_k$ , then  $MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk})$ .

622 When a relevant document is not retrieved at all, the pre-  
 623 cision value in the above equation is taken to be 0.

624 **Mean reciprocal rank (MRR):** The reciprocal rank of a  
 625 query response is the multiplicative inverse of the rank of  
 626 the first correct answer. The mean reciprocal rank (MRR) is  
 627 the average of the reciprocal ranks of results for a sample of  
 628 queries  $Q$ :  $MRR(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{rank_j}$ , where  $rank_j$  refers to  
 629 the rank position of the first relevant document for the  $j$ -th  
 630 query.

### 631 3.3 Comparison Baselines

632 In order to validate the effectiveness of the proposed  
 633 approach, we evaluated 8 models in our experiments and  
 634 provided the corresponding results as baselines.  
 635

- 636 • **CnBM25:** This is the baseline model which selects  
 637 candidate software documents by content described  
 638 in Section 2.2.1. Note that this model indexes each  
 639 software document with the content stated in itself  
 640 and retrieves candidates using BM25 [42] scoring  
 641 function<sup>13</sup>.
- 642 • **CnLDA:** This baseline is also based on the content  
 643 of software documentation in Section 2.2.1. This model  
 644 represents query and content of documentation  
 645 as vectors in topic space. In our experiments, we  
 646 use Latent Dirichlet Allocation (LDA) [23] model  
 647 for topic modeling. In particular, we computed the  
 648 perplexity [23] of a held-out test set of training data  
 649 to determine the number of topics. The perplexity is  
 650 monotonically decreasing in the likelihood of the test

651 data, and is algebraically equivalent to the inverse of  
 652 the geometric mean per-word likelihood. We choose  
 653 the number of topics (*i.e.*, 200) with the lowest per-  
 654 plexity.

- 655 • **CnDoc2vec:** This baseline [48] is based on the  
 656 doc2vec approach proposed by Google, which con-  
 657 catenates the paragraph vector with several word  
 658 vectors from a paragraph and predicts the following  
 659 word in the given text. The document vectors are  
 660 learned from the content of software documentation.
- 661 • **CxBM25:** This baseline is based on the local context  
 662 of software documentation described in Section 2.2.2.  
 663 This model indexes each local context as a docu-  
 664 ment and retrieves candidates using BM25 scoring  
 665 function. Given a query, this method is to retrieve the  
 666 most similar context and find out official software  
 667 documents in the best answer as the recommenda-  
 668 tion.
- 669 • **CxLDA:** This baseline is similar with **CnLDA**. The  
 670 difference is that this baseline uses the topic space of  
 671 local context for query and documentation represen-  
 672 tation. The lowest perplexity results in 300 topics for  
 673 LDA models.
- 674 • **LoCxDoc2vec:** This baseline is similar with **Cn-**  
 675 **Doc2vec**, but the document vectors are learned from  
 676 local contexts.
- 677 • **GloCxCos:** This baseline is base on the global con-  
 678 text describe in Section 2.2.3. This model represents  
 679 natural language words and software documents as  
 680 vectors in shared embedding space [28]. The cosine  
 681 similarity is used for retrieval.
- 682 • **GloCxAsy:** This baseline is based the global context  
 683 and an asymmetric similarity measure [49] is used  
 684 to retrieve candidates.
- 685 • **CnCxL2R:** This method is our proposed model which  
 686 takes into account content, local and global con-  
 687 text of software documentation in a learning-to-rank  
 688 schema as shown in Fig. 2.

## 689 4 EXPERIMENTAL RESULTS

690 In this section, we present our experimental results. First,  
 691 we compare the performance of different models described  
 692 in Section 3.3 and report the performance of different feature  
 693 groups. Second, we present the effect of embedding dimen-  
 694 sionality. Finally, we present a comparative result against  
 695 Google Search.  
 696

### 697 4.1 Performance of Different Models

698 In this section, we compare our model CnCxAPI with  
 699 other 8 baseline models. Table 2 shows the recommendation  
 700 performance by using different evaluation metrics discussed  
 701 in Section 3.2. We have the following observations:

- 702 (1) From Table 2, we note that the performance of  $P@k$   
 703 of CnCxL2R significantly outperforms all the remaining  
 704 baseline models for all values of  $k$  ( $k = 1, 2, 5, 10$ ) in our  
 705 dataset. We also conduct a statistical  $t$ -test and the results  
 706 show that the improvements between the CnCxAPI model  
 707 and the other 8 baselines are statistically significant

13. We use the BM25 implementation provided by Apache Lucene.



TABLE 2

Performance ( $P@k$ ,  $MAP$  and  $MRR$ ) for different models. The bold format indicates the best performance. † indicates that the difference between the results of  $CnCXL2R$  and other models are significant with  $p < 0.05$  under  $t$ -test.

Model	$P@1$	$P@2$	$P@5$	$P@10$	$R@1$	$R@2$	$R@5$	$R@10$	$MAP$	$MRR$
CnBM25	0.1334	0.1073	0.0743	0.0510	0.1128	0.1766	0.3021	0.4089	0.2214	0.2248
CnLDA	0.0854	0.0672	0.0435	0.0317	0.0789	0.1028	0.2343	0.3562	0.1637	0.1729
CnDoc2vec	0.1172	0.0879	0.6725	0.0487	0.9216	0.1321	0.2745	0.3698	0.1893	0.2014
LoCxBM25	0.1790	0.1461	0.1018	0.0719	0.1482	0.2368	0.4066	0.5641	0.2969	0.3032
LoCxLDA	0.1225	0.0942	0.0743	0.0519	0.0996	0.1508	0.2946	0.4029	0.2075	0.2107
LoCxDoc2vec	0.1341	0.1012	0.0677	0.0532	0.1084	0.1833	0.3278	0.4237	0.2368	0.2435
GloCxCos	0.1517	0.1228	0.0847	0.0607	0.1276	0.2022	0.3430	0.4817	0.2522	0.2575
GloCxAsy	0.1811	0.1523	0.1087	0.0834	0.1646	0.2573	0.4394	0.6233	0.3153	0.3364
CnCXL2R	<b>0.2238†</b>	<b>0.1766†</b>	<b>0.1230†</b>	<b>0.0904†</b>	<b>0.1842†</b>	<b>0.2862†</b>	<b>0.4853†</b>	<b>0.7043†</b>	<b>0.3779†</b>	<b>0.3868†</b>

( $p < 0.05$ ). In particular,  $P@10$  is very small because it is calculated by  $P@10 = \frac{|D_{10} \cap D_g|}{10}$ , where  $D_{10}$  is the set of top-10 recommended software documents and  $D_g$  is the set of ground-truth documents. For most of discussion threads on Stack Overflow, the best answers commonly contain only one software document, so  $|D_{10} \cap D_g| = 1$  in most cases. Thus, the ideal value of  $P@10$  is slightly above 0.1. Although  $P@10$  of CnCXL2R is very small (*i.e.*, 0.0904), it almost reaches the ideal value.

(2) The performance of  $R@k$  of CnCXL2R significantly outperforms the other 8 baselines for all values of  $k$  ( $k = 1, 2, 5, 10$ ). It is worth noting that CnCXL2R achieves highest recall (0.7043) at  $k = 10$ .

(3) We can also observe that the performance of  $MAP$  and  $MRR$  of CnCXL2R significantly outperform all other methods. The  $MAP$  of CnCXL2R is equal to 0.3779, which achieves 70.69%, 130.84%, 99.63%, 27.28%, 82.12%, 59.58%, 49.84%, and 20.54% relative improvements over CnBM25, CnLDA, CnDoc2vec, LoCxBM25, LoCxLDA, LoCxDoc2vec, GloCxCos and GloCxAsy, respectively. Likewise, the performance of  $MRR$  of CnCXL2R achieves relative improvements with 72.06%, 123.71%, 92.05%, 27.57%, 83.57%, 58.85%, 50.21%, and 14.98%, respectively.

(4) The context-based model outperforms corresponding content-based model (*e.g.*, LoCxBM25 better than CnBM25). This observation confirms that there exists an information gap between the intent in queries and the implementation in software documentation.

(5) Another observation is that BM25-based models (CnBM25 and LoCxBM25) achieve better performance than topic-based models (CnLDA and LoCxLDA). The reason for this difference may be that the length of query is short while the length of software documentation is often long. In such case, the query is not effectively represented in topic space. Thus it leads to poor performance when retrieving software documents using distributed topics of a query.

## 4.2 Performance of Different Feature Combinations

As shown in Table 1, we divide the 22 features into four groups and three categories. In this Section, we investigate the performance of different feature combinations in the learning-to-rank framework.

As shown in Table 3, we totally produce eight feature combinations in terms of feature groups and categories. The first combination considers all of the 22 features in the learning-to-rank schema. The combinations (*i.e.*, #2 – #8) consider all features excluding context features, textual

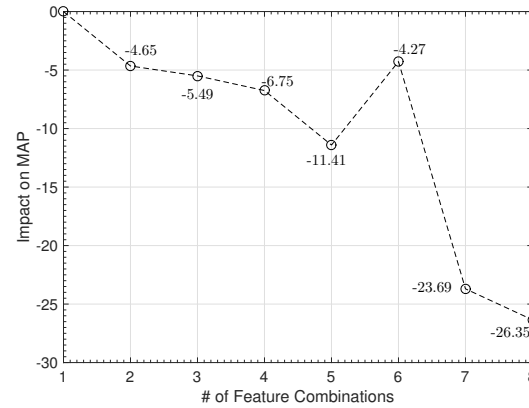


Fig. 6. The impact of feature combinations on MAP.

features, statistical-based features, popularity features, Q-class features, D-class features and QD-class features, respectively. The results show that the improvements between the first combination and the other seven combinations are statistically significant at  $p < 0.05$ , in terms of  $MAP$ ,  $MRR$ ,  $P@1$ ,  $P@10$ ,  $R@1$  and  $R@10$ . These results confirm that these features derived from official documentation and social context, are effective in recommending software documentation for developers.

We consider the first combination (*i.e.*, #1 in Fig. 6) as the baseline. Then we define the performance impact as  $\frac{P_n - P_1}{P_1} \times 100\%$ , where  $P_1$  is the performance metric of the first combination, and  $P_n$  ( $n = 2, \dots, 8$ ) is the performance metric from the second to eighth combination. Fig. 6 shows the impact of different feature combinations on  $MAP$ . We can observe that the second feature combination (excluding context features) decreases the performance of  $MAP$  by  $-4.65\%$  compared with the first combination (all features), and followed by  $-5.49\%$ ,  $-6.75\%$ ,  $-11.41\%$ ,  $-4.27\%$ ,  $-23.69\%$ , and  $-26.35\%$  for the feature combination #2 to #8, respectively. Among the combinations of #2, #3, #4, and #5, the performance of #5 (excluding popularity features) is worst with impact of  $-11.41\%$ . The sixth combination (excluding query-dependent features) has little effect with impact of  $-4.27\%$ . On the contrary, the seventh combination (excluding document-dependent features) and eighth combination (excluding query-document dependent features) have influential effect on  $MAP$  with impact of  $-23.69\%$  and  $-26.35\%$ , respectively.

## 4.3 Effect of Embedding Dimensionality

The dimensionality of word embedding is an impor-

TABLE 3

Performance of different feature combinations. The bold format indicates the best performance. † indicates that the difference between the results of the first combination and other combinations are significant with  $p < 0.05$  under  $t$ -test.

#	Feature Combination	MAP	MRR	P@1	P@10	R@1	R@10
1	all features	<b>0.3779†</b>	<b>0.3868†</b>	<b>0.2238†</b>	<b>0.0904†</b>	<b>0.1842†</b>	<b>0.7043†</b>
2	all features excluding context features	0.3611	0.3763	0.2104	0.0814	0.1739	0.6110
3	all features excluding textual features	0.3582	0.3747	0.2089	0.0886	0.1705	0.6868
4	all features excluding statistical-based features	0.3540	0.3700	0.2044	0.0895	0.1669	0.6950
5	all features excluding popularity features	0.3392	0.3533	0.1981	0.0803	0.1674	0.6281
6	all features excluding Q-class features	0.3624	0.3791	0.2172	0.0799	0.1792	0.6905
7	all features excluding D-class features	0.3055	0.3202	0.1700	0.0747	0.1394	0.5854
8	all features excluding QD-class features	0.2991	0.3098	0.1374	0.0854	0.1124	0.6565

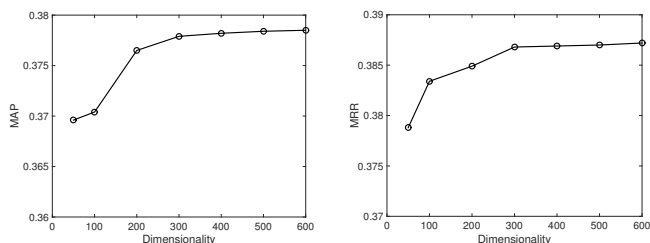


Fig. 7. The effect of different dimensionality of word embedding for learning-to-rank.

tant factor for embedding performance. Furthermore, the performance of word embedding directly affects candidate documentation selection and the context similarity feature listed in Table 1. Thus, the word embedding can affect the final ranking performance in many ways. When we train the word representations, we need specify the parameter of number of dimensionality. In this Section, we investigate the effect of embedding dimensionality with values 50, 100, 200, 300, 400, 500, and 600.

Fig. 7 shows the effect of different dimensionality on learning-to-rank schema in terms of  $MAP$  and  $MRR$ . We observe that the increase of embedding dimensionality from 50 to 300, leads to a gradual improvement in performance. The optimal performance can be achieved at the dimensionality of 300. After 300 dimensions, the performance is not improved significantly, while the training procedure becomes extremely slow. Thus in this research, we set 300 as the value of the dimensionality.

#### 4.4 Comparison with Google Search

Given a natural language query, the search engine of Stack Overflow can only return a list of discussion threads and could not directly retrieve software documentation. However, Google search can return a list of software documentation. Moreover, Google search is the most popular help-seeking tool when develops need to search for solutions for programming problems. Thus, we compare the results of CnCxL2R with the Google search engine.

##### 4.4.1 Setup

Among the 2924 testing discussion threads, we randomly select 30 discussion threads and consider the titles of these discussion threads as queries. Note that the 30 queries are not used in the training set and are listed in Table 4. Given a natural language query, we first get a list of web pages via Google search engine. As we focus on the official documentation in this research, we restrict the retrieved results through appending official sites in the queries. For

example, the first query in Table 4 was performed with “get objects from a BlockingQueue site:docs.oracle.com/javase” using Google search engine in August, 2016. For each query in Table 4, we get the top-10 software documents by the Google search engine and CnCxL2R separately.

We use three metrics [16], [50] to measure the performance of the two list of software documents. The metric  $FR$  represents the rank of the first relevant software documentation in the results list of a query. This metric is important as most users scan the results from top to bottom. The smaller the number of  $FR$ , the better the performance. The metric  $RR5$  denotes the relevancy ration of software documents in the top 5 results. The metric  $RR10$  represents the relevancy ration of software documents in the top 10 results.  $RR5$  and  $RR10$  range from 0 to 100. The higher the values of  $RR5$  and  $RR10$ , the better the performance.

We recruited two developers who both are postdoctoral fellows with 4-years+ programming experiences in Java, to manually check the two lists generated by Google search engine and CnCxL2R. Each software documentation in candidate list was marked relevant or irrelevant, indicating whether the developer considered this software documentation is related to the task in the query. The two developers separately annotated the top 10 software documents for each evaluated query and then we combine the results. If any inconsistency found while combining the results, the developers discussed among them and then remarked until a consensus is reached. The Cohen’s kappa coefficient is equal to 0.846, which shows that the two annotators have a good agreement amongst themselves.

##### 4.4.2 Comparison Results

Table 4 shows the performance comparison of Google search and CnCxL2R. In particular, the symbol “-” in second column indicates that there is no relevant software documentation returned by Google search engine for the query. The last row shows the average performance in terms of the defined three metrics.

Compared to  $FR$  of Google search, CnCxL2R achieves better performance with average  $FR$  of 1.3. For most of the queries (23 out of 30 queries), CnCxL2R is able to recommend relevant software documentation at the first position in the result list. On the contrary, Google search is able to handle only 11 queries in which the first returned software documentation is relevant to the given query. Specifically, for 5 out of 30 queries, Google search could not return any relevant software documentation in top 10 results. The  $p$ -value of  $FR$  comparison shows that there is a

TABLE 4

Comparison between Google search and CnCXL2R (*FR*: the rank of the first relevant documentation. *RR5*: top 5 relevancy ratio. *RR10*: top 10 relevancy ratio. “java.xxx.xxx” indicates Java API documentation. “specs\_xxx/xxx” indicates Java language specification. “tutorial/xxx/xxx” indicates Java tutorial. † indicates the differences between CnCXL2R and Google are significant with  $p < 0.05$  under *t*-test.)

query	Google Search			CnCXL2R			Top 3 relevant documents by CnCXL2R
	<i>FR</i>	<i>RR5</i>	<i>RR10</i>	<i>FR</i>	<i>RR5</i>	<i>RR10</i>	
get objects from a BlockingQueue	1	20	60	1	40	80	java.util.concurrent.BlockingQueue; java.util.concurrent.ArrayBlockingQueue; java.util.concurrent.ConcurrentLinkedQueue
java string split inconsistency	6	0	20	1	60	70	java.lang.String.split(); java.lang.String.substring(); java.lang.String.contains()
create a Series using Iterators	2	40	20	2	60	40	java.util.Iterator; java.util.Iterable; java.util.ListIterator
rules governing narrowing of double to int	1	20	30	1	60	60	specs_5.1.3. Narrowing Primitive Conversion; specs_3.10.2. Floating-Point Literals; specs_15.25. Conditional Operator ? :
add submenu to MenuItem	1	80	70	2	40	20	tutorial/uiswing/components/dialog.html; tutorial/uiswing/components/menu.html; tutorial/uiswing/layout/visual.html
show a timer in a Java Frame	1	40	30	1	60	50	tutorial/uiswing/misc/timer.html; java.util.Timer; javax.swing.Timer
create utility to cast objects to beans	5	20	10	4	40	40	java.lang.Class.cast(); java.lang.Object.getClass(); java.lang.Class.getResourceAsStream(); java.io.BufferedReader.readLine(); java.io.InputStream.read();
java Client/Server App will not readLine()	-	0	0	1	60	80	tutorial/networking/sockets/clientServer.html
synchronize concurrent collections	1	20	10	1	100	90	java.util.concurrent.ConcurrentHashMap; java.util.concurrent.ConcurrentLinkedQueue; java.util.Collections.synchronizedList()
pull double out of string with matcher	-	0	0	1	80	70	java.util.regex.Matcher; java.util.regex.Pattern; java.lang.String.matches()
java hashCode() collision for objects	-	0	0	1	60	50	java.lang.Object.hashCode(); java.lang.String.hashCode(); java.lang.System.identityHashCode()
java reverse string	1	60	50	2	60	40	java.lang.StringBuilder; java.lang.String.format(); java.lang.String.compareTo()
get char from multi-string array	3	40	30	1	100	60	java.lang.String.charAt(); java.lang.String.split(); java.lang.String.replace()
search for the index of an element in an array	1	40	30	1	80	50	java.util.Arrays.binarySearch(); tutorial/java/nutsandbolts/arrays.html; java.util.ArrayList.get()
replace last space in a string	2	40	20	1	100	80	java.lang.String.trim(); java.lang.String.split(); java.lang.String.indexOf()
different font for combo box in Java	4	20	10	1	60	40	tutorial/uiswing/components/combobox.html#renderer; java.awt.Font; tutorial/uiswing/components/combobox.html
add days to date in Java	4	20	30	1	100	70	java.text.SimpleDateFormat; api.java.util.Date; java.text.DateFormat
support println in a class	1	20	10	1	100	70	java.io.PrintStream.println(); api.java.lang.Object; java.lang.System.setOut()
remove row from table	1	40	30	1	80	70	javax.swing.table.DefaultTableModel; javax.swing.JTable; javax.swing.table.AbstractTableModel
primitive cast and assignments in Java	2	40	30	1	60	60	spec_5.2. Assignment Contexts; spec_3.10.2. Floating-Point Literals; spec_5.5. Casting Contexts
draw points as ovals using java swing	1	60	60	1	100	80	tutorial/uiswing/painting/index.html; javax.swing.JComponent.paintComponent(); tutorial/uiswing/concurrency/dispatch.html
julian date to regular date conversion	2	20	10	1	60	80	java.text.SimpleDateFormat; api.java.util.Date; java.text.DateFormat
reinitialise transient variable	-	0	0	3	40	60	java.io.Serializable; specs_15.28. Constant Expressions; java.util.Scanner
invalid class exception: no valid constructor	5	20	20	1	40	50	java.io.Serializable; java.lang.Class.newInstance(); java.lang.IllegalArgumentException
combine explicit locks with synchronized methods	1	100	50	1	80	70	java.util.concurrent.locks.ReentrantLock; java.util.concurrent.locks.Lock; tutorial/essential/concurrency/locksync.html
parse a number from a string	2	40	40	3	40	70	java.text.SimpleDateFormat; java.lang.Integer.parseInt(); java.text.NumberFormat.parse()
format double with zeros on left and right side	-	0	0	1	60	70	java.text.DecimalFormat; tutorial/uiswing/layout/visual.html; api.java.util.Formatter
wait for a key to be pressed inside loop	4	20	20	1	60	60	tutorial/uiswing/misc/keybinding.html; tutorial/uiswing/concurrency/dispatch.html; tutorial/uiswing/events/keylistener.html
get single bytes from multi-byte variable in java	3	20	20	1	40	50	java.nio.ByteBuffer; java.lang.String.String(); spec_5.2. Assignment Contexts
change PWD of linux from JSP	-	0	0	2	40	60	java.lang.Runtime.exec(); java.lang.ProcessBuilder.directory(); java.util.ResourceBundle
average	> 2.3	28	24	1.3†	65†	61†	

871 statistical significance of the improvement of CnCXL2R over  
 872 Google search. Compared to the Google search (average  
 873 *RR5* is 28 and average *RR10* is 24), CnCXL2R achieves  
 874 much higher average value for *RR5* (65) and *RR10* (61).  
 875 Therefore, CnCXL2R can recommend much more relevant  
 876 software documents in top 10 results than Google search.  
 877 The differences between these two approaches in terms of  
 878 *RR5* and *RR10* are statistically significant at  $p < 0.05$ .

879 The last column shows the top three relevant docu-  
 880 ments recommended by CnCXL2R, which consist of Java  
 881 API documents, Java language specifications and Java tu-  
 882 torials. We observe that CnCXL2R can more effectively  
 883 respond to bug-like queries over Google search. For in-  
 884 stance, Google search could not return relevant API do-  
 885 cumentation given the query “*java Client/Server App will*  
 886 *not readLine()*”, but CnCXL2R can recommend eight re-  
 887 levant API documents in top 10 results. More impor-  
 888 tantly, the first recommended software documentation (  
 889 `java.io.BufferedReader.readLine()`) is extremely  
 890 relevant to the query. Likewise, CnCXL2R can recommend  
 891 five relevant software documents whereas the Google search  
 892 can only return one document with respect to the exception  
 893 query “*invalid class exception: no valid constructor*”.

894 Another observation is that CnCXL2R can effectively  
 895 respond to those queries which do not contain API-like  
 896 words. For example, the query “*format double with zeros on*  
 897 *left and right side*” does not explicitly contain API-like words.  
 898 Thus, Google search cannot effectively handle this case  
 899 and fails to return any relevant software documentation.  
 900 However, CnCXL2R can recommend seven high quality soft-  
 901 ware documents including `java.text.DecimalFormat`  
 902 and `api.java.util.Formatter`. In the same way, for  
 903 the query “*reinitialise transient variable*” and “*change PWD*

of linux from JSP”, CnCXL2R can recommend extremely re- 904  
 levant software documents `java.io.Serializable` and 905  
`java.lang.Runtime.exec()`, respectively. In such cases, 906  
 CnCXL2R can successfully bridge the lexical gap and iden- 907  
 tify the semantics between the query and the documenta- 908  
 tion. 909

910 However, CnCXL2R cannot recommend good software  
 911 documentation for some queries. For example, for the query  
 912 “*create utility to cast objects to beans*”, CnCXL2R cannot recom-  
 913 mend any documentation about “*beans*”. After manually  
 914 checking our training dataset, we find that there is not  
 915 enough context about the topic “*beans*”. This causes the  
 916 coverage of this query incomplete. In the future, we will  
 917 collect more context of different topics and improve the  
 918 coverage of our training dataset accordingly. 918

## 5 THREATS TO VALIDITY 919

920 The automatically annotated data is one of the threats to  
 921 validity of our experiment. We built the ground-truth da-  
 922 taset via judging software documents in the best answers  
 923 in Section 4.1. For some cases, the recommended software  
 924 documents which are not in the best answer, may be useful  
 925 for the issued query. It was a tedious task to annotate the  
 926 ground truth of the 14, 944 discussion threads in our corpus.  
 927 However, our human evaluation provides a supplementary  
 928 study about this threat. The experiment setting of automatic  
 929 evaluation makes the  $P@k$  very low, but CnCXL2R almost  
 930 reaches the ideal value. Even so, the high values of  $R@k$   
 931 indicate the validity of CnCXL2R. 931

932 In Section 4.4, we employed two annotators to label the  
 933 two recommended lists generated by Google search and  
 934 CnCXL2R. This is a threat to validity because the judgments 934

are based on the knowledge background of the two annotators. For a natural language query about programming task, there are many approaches to implement the task with different APIs. To make it more reliable, we require the two annotators to discuss until consensus is reached when a judgment is inconsistent for a query. The relevant software documents reflect the most common implementations for the programming task in a query. We believe that the discussion can eliminate the threat to some extent.

Another threat is the training data. In our approach, the local context and global context are from the discussion threads on Stack Overflow. Most of the contexts are free-form texts posted by the users of Stack Overflow, rather than the quotation of official documentation. For the free-form texts posted by users, there may be spelling mistakes and typos. However, the quality of the best answers is better than other answers on Stack Overflow [51]. Thus, we restrict the context in best answers and we believe it can eliminate the threat to some extent.

Last but not least, the coverage of the training data is a threat to validity. In this study, we only test the performance of CnCXL2R on Java documentation and collect the discussion threads about Java topic on Stack Overflow. The coverage of the training dataset potentially affects the query response performance. In this study, the performance on Java related programming task is reasonable. It is still uncertain how CnCXL2R will perform on other datasets. In future, we will perform more evaluations on different training sets and create a better training set.

## 6 RELATED WORK

In this Section, we present a literature review on mining software documentation, divided into API usage, API recommendation and harnessing Stack Overflow data.

### 6.1 API usage and API Recommendation

There is a number of techniques which mine API usage [52], [53] and recommend APIs from the perspectives of the content of documentation and code corpus.

API usage patterns are frequent API method call sequences. Xie *et al.* [52] presented MAPO, which mined API usages from open source repositories based on existing source code search engines. Given a query consisting of method names or class names, MAPO can generate a list of method call sequences. Wang *et al.* [53] developed API usage pattern miner (UP-Miner) to mine succinct and high-coverage API usage patterns from source code. Given a user-specified API method, UP-Miner returns code snippets as reuse candidates via mining frequent closed API-method invocation sequences. Acharya *et al.* [54] developed a framework to automatically extract frequent partial orders among user-specified APIs, assisting effective API reuse and checking. Moritz *et al.* [55] presented Export, which can automatically mine and visualize API usages in large source code repositories. Given a starting API, Export can recommend complex API usage examples from a large repository.

Recently, Pham *et al.* [56] proposed an approach to learn API usages from bytecode of Android mobile applications, which uses Hidden Markov Model to represent method

TABLE 5

Comparison of our work with other related approaches. Column *Information* specifies that which information is used in each approach (Source Code (SC), Content of Documentation (CN), Context on Stack Overflow (CX)). Column *Input Type* specifies the input format for each approach. Column *Output Type* specifies the output format for each approach.

Approach	Information	Input Type	Output Type
MAPO [52]	SC	API name	API usage
UP-Miner [53]	SC	API name	API usage
ExPort [55]	SC	API name	API usage
SWIM [50]	SC	API-like query	API usage
DeepAPI [16]	SC	API-like query	API usage
XSnippet [60]	SC	Code snippet	Code snippet
Baker [4]	CN	Code snippet	Software documentation
Krec [17]	CN	Code snippet	Software documentation
TaskNavigator [11]	CN	Semi-free-form	Section fragment
ROSF [61]	SC	Free-form	Code snippet
Portfolio [57]	SC	Free-form	API usage
RACK [15]	CX	Free-form	API class
CnCXL2R (our)	CN, CX	Free-form	Software documentation

call sequences. Raghothaman *et al.* [50] presented SWIM to capture API usage patterns based on the defined call sequences. SWIM is based on open-source code repositories and clickthrough data from the Bing search engine. Gu *et al.* [16] developed DeepAPI, a deep learning approach to capture API usage sequences using the Recurrent Neural Network (RNN) encoder-decoder models. Compared to SWIM, DeepAPI can recommend more accurate API usage sequences.

McMillan *et al.* [57] developed Portfolio to recommend and visualize relevant APIs and their usages for code search. Their experiments show that Portfolio can find more relevant APIs compared to Google Code Search. Given simple text phrases, Chan *et al.* [58] proposed subgraph search approach for API recommendation via modeling API invocations as an API graph. Thung *et al.* [59] developed a technique which recommends API methods for a query of textual description of a feature request.

Recently, Subramanian *et al.* [4] developed a constraint-based approach to identify fine-grained type references, method calls, and field references in source code snippets. This approach can correctly link source code examples to official API documentation. Robillard *et al.* [17] developed a technique to automatically detect knowledge items and extract word patterns in software documentation. They use these patterns to recommend reference API documentation for code fragments. Rahman *et al.* [15] proposed RACK, using regular expressions to extract API class and to build keyword-API associations from the posts on Stack Overflow. Given some code search keywords, RACK can recommend a list of relevant API classes. This work is close to our work, but it is based on Java class level instead of Java method level. Moreover, it did not exploit the content of official documentation instead of using keyword-API co-occurrence of social context.

Different from the existing work from the view points of program analysis and mining code repositories, our approach exploits the official content of documentation and high-quality social context on Stack Overflow for recommendation. A relative comparison of our approach with the existing approaches is presented in Table 5.

## 6.2 Harnessing Stack Overflow Data

The Stack Overflow data has attracted much research interest [5], [6] in recent years. Some tools were developed for assisting software development. Bacchelli *et al.* [62] developed Seahawk, an Eclipse plugin that can automatically integrate crowd knowledge of Stack Overflow into the Integrated Development Environment (IDE). This tool brings the convenience for developers that they can directly access Stack Overflow data without switching their work context. Ponzanelli *et al.* [63] presented Prompter, a self-confident recommender system that automatically searches and identifies relevant Stack Overflow discussions under the code context in the IDE. San Pedro *et al.* [64] proposed Rank-SLDA, recommending questions for collaborative Q&A systems based on developers' topics of expertise. Cordeiro *et al.* [65] developed a tool, recommending question answering web resources in IDE based on the information of exception stack traces. Treude *et al.* [10] presented SISE, automatically augmenting API documentation with "insight sentences" from Stack Overflow.

In addition, some researchers have contributed their efforts for program comprehension using Stack Overflow data. The study [66] revealed that their tool and Stack Overflow data are capable of sometimes coming up with surprising insights that aid a developer both for program comprehension and software development. Treude *et al.* [67] investigated how developers ask and answer questions on the Web. Linares-Vásquez *et al.* [68] took an empirical study on how do API changes trigger Stack Overflow discussions. Recently, Nadi *et al.* [3] performed an empirical investigation into the obstacles developers face while using the Java cryptography APIs based on 100 StackOverflow posts, 100 GitHub repositories, and survey input from 48 developers.

In contrast to these work, the goal of our approach is to bridge the information gap between natural language query and official software documentation harnessing Stack Overflow data.

## 7 CONCLUSION AND FUTURE WORK

Traditional code search engines and existing API recommendation systems often do not perform well with natural language queries, especially which do not contain API-like terms. In this paper, we attempt to leverage official content and social context to recommend software documentation. Our proposed solution, named CnCxl2R, exploits the official content and social context in a learning-to-rank schema. We identify 22 features to learn a ranker. We conduct a set of experiments to evaluate the effectiveness of our approach. The results of these experiments show that CnCxl2R outperforms baseline models and is effective for natural language queries.

As future work, we plan to create a better training set with higher coverage for improving the performance of recommendation. As human feature engineering consumes massive manpower, we will also investigate the automatic feature engineering in our approach using deep neural network.

## REFERENCES

- [1] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *Proc. ICSE*, 2007, pp. 344–353.
- [2] A. J. Ko, B. A. Myers, M. J. Coblentz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE Trans. Softw. Eng.*, vol. 32, no. 12, pp. 971–987, 2006.
- [3] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, "Jumping through hoops: why do java developers struggle with cryptography apis?" in *Proc. ICSE*, 2016, pp. 935–946.
- [4] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live api documentation," in *Proc. ICSE*, 2014, pp. 643–652.
- [5] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empir. Softw. Eng.*, vol. 19, no. 3, pp. 619–654, 2014.
- [6] A. Anderson, D. Huttenlocher, J. Kleinberg, and J. Leskovec, "Discovering value from community activity on focused question answering sites: a case study of stack overflow," in *Proc. KDD*, 2012, pp. 850–858.
- [7] B. Vasilescu, V. Filkov, and A. Serebrenik, "Stackoverflow and github: Associations between software development and crowdsourced knowledge," in *Proc. SocialCom*, 2013, pp. 188–195.
- [8] M. P. Robillard and R. Deline, "A field study of api learning obstacles," *Empir. Softw. Eng.*, vol. 16, no. 6, pp. 703–732, 2011.
- [9] B. Dagenais and M. P. Robillard, "Recovering traceability links between an api and its learning resources," in *Proc. ICSE*, 2012, pp. 47–57.
- [10] C. Treude and M. P. Robillard, "Augmenting api documentation with insights from stack overflow," in *Proc. ICSE*, 2016, pp. 392–403.
- [11] C. Treude, M. P. Robillard, and B. Dagenais, "Extracting development tasks to navigate software documentation," *IEEE Trans. Softw. Eng.*, vol. 41, no. 6, pp. 565–581, 2015.
- [12] P. C. Rigby and M. P. Robillard, "Discovering essential code elements in informal documentation," in *Proc. ICSE*, 2013, pp. 832–841.
- [13] G. Petrosyan, M. P. Robillard, and R. De Mori, "Discovering information explaining api types using text classification," in *Proc. ICSE*, 2015, pp. 869–879.
- [14] H. Jiang, J. Zhang, X. Li, Z. Ren, and D. Lo, "A more accurate model for finding tutorial segments explaining apis," in *Proc. SANER*, vol. 1, 2016, pp. 157–167.
- [15] M. M. Rahman, C. K. Roy, and D. Lo, "Rack: Automatic api recommendation using crowdsourced knowledge," in *Proc. SANER*, vol. 1, 2016, pp. 349–359.
- [16] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep api learning," 2016.
- [17] M. P. Robillard and Y. B. Chhetri, "Recommending reference api documentation," *Empir. Softw. Eng.*, vol. 20, no. 6, pp. 1558–1586, 2015.
- [18] H. Niu, I. Keivanloo, and Y. Zou, "Learning to rank code examples for code search engines," *Empir. Softw. Eng.*, pp. 1–33, 2016.
- [19] E. C. Campos, L. B. de Souza, and M. d. A. Maia, "Searching crowd knowledge to recommend solutions for api usage tasks," pp. 1–32, 2016.
- [20] G. Zhou, Y. Zhou, T. He, and W. Wu, "Learning semantic representation with neural networks for community question answering retrieval," *Knowl.-Based Syst.*, vol. 93, pp. 75–83, 2016.
- [21] S. Bird, "Nltk: the natural language toolkit," in *Proc. COLING/ACL*, 2006, pp. 69–72.
- [22] P. Willett, "The porter stemming algorithm: then and now," *Program*, vol. 40, no. 3, pp. 219–223, 2006.
- [23] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, no. Jan, pp. 993–1022, 2003.
- [24] Z. S. Harris, "Distributional structure," in *Pap. struct. transform. linguist.*, 1970, pp. 775–794.
- [25] M. Grbovic, N. Djuric, V. Radosavljevic, F. Silvestri, and N. Bhamidipati, "Context-and content-aware embeddings for query rewriting in sponsored search," in *Proc. SIGIR*, 2015, pp. 383–392.
- [26] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, no. Aug, pp. 2493–2537, 2011.
- [27] J. Turian, L. Ratinov, and Y. Bengio, "Word representations: a simple and general method for semi-supervised learning," in *Proc. ACL*, 2010, pp. 384–394.
- [28] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[29] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. NIPS*, 2013, pp. 3111–3119.

[30] T. Van Nguyen, A. T. Nguyen, and T. N. Nguyen, "Characterizing api elements in software documentation with vector representation," in *Proc. ICSE*, 2016, pp. 749–751.

[31] D. H. Dalip, M. A. Gonçalves, M. Cristo, and P. Calado, "Exploiting user feedback to learn to rank answers in q&a forums: a case study with stack overflow," in *Proc. SIGIR*, 2013, pp. 543–552.

[32] P. Li, Q. Wu, and C. J. Burges, "Mcrank: Learning to rank using multiple classification and gradient boosting," in *Proc. NIPS*, 2007, pp. 897–904.

[33] L. Li and H.-T. Lin, "Ordinal regression by extended binary classification," in *Proc. NIPS*, 2006, pp. 865–872.

[34] W. S. Cooper, F. C. Gey, and D. P. Dabney, "Probabilistic retrieval based on staged logistic regression," in *Proc. SIGIR*, 1992, pp. 198–210.

[35] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma, "Frank: a ranking method with fidelity loss," in *Proc. SIGIR*, 2007, pp. 383–390.

[36] T. Pahikkala, E. Tsivtsivadze, A. Airola, J. Boberg, and T. Salakoski, "Learning to rank with pairwise regularized least-squares," in *Proc. SIGIR*, vol. 80, 2007, pp. 27–33.

[37] Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen, and G. Sun, "A general boosting method and its application to learning ranking functions for web search," in *Proc. NIPS*, 2008, pp. 1697–1704.

[38] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *Proc. ICML*, 2007, pp. 129–136.

[39] M. N. Volkovs and R. S. Zemel, "Boltzrank: learning to maximize expected ranking gain," in *Proc. ICML*, 2009, pp. 1089–1096.

[40] H. Valizadegan, R. Jin, R. Zhang, and J. Mao, "Learning to rank by optimizing ndcg measure," in *Proc. NIPS*, 2009, pp. 1883–1891.

[41] T. Qin, T.-Y. Liu, J. Xu, and H. Li, "Letor: A benchmark collection for research on learning to rank for information retrieval," *J. Inf. Retr.*, vol. 13, no. 4, pp. 346–374, 2010.

[42] S. E. Robertson, S. Walker, S. Jones *et al.*, "Okapi at trec-3," 1995.

[43] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao, "Adapting boosting for information retrieval measures," *J. Inf. Retr.*, vol. 13, no. 3, pp. 254–270, 2010.

[44] C. Quoc and V. Le, "Learning to rank with nonsmooth cost functions," *Proc. NIPS*, vol. 19, pp. 193–200, 2007.

[45] C. J. Burges, "From ranknet to lambdamart to lambdamart: An overview," *Learning*, vol. 11, pp. 23–581, 2010.

[46] R. Baeza-Yates, B. Ribeiro-Neto *et al.*, *Modern information retrieval*, 1999, vol. 463.

[47] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," *ACM Trans. Inf. Syst.*, vol. 20, no. 4, pp. 422–446, 2002.

[48] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proc. ICML*, 2014, pp. 1188–1196.

[49] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu, "From word embeddings to document similarities for improved information retrieval in software engineering," in *Proc. ICSE*, 2016, pp. 404–415.

[50] M. Raghathan, Y. Wei, and Y. Hamadi, "Swim: Synthesizing what i mean," in *Proc. ICSE*, 2016.

[51] E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne, "Finding high-quality content in social media," in *Proc. WSDM*, 2008, pp. 183–194.

[52] T. Xie and J. Pei, "Mapo: Mining api usages from open source repositories," in *Proc. MSR*, 2006, pp. 54–57.

[53] J. Wang, Y. Dang, H. Zhang, K. Chen, T. Xie, and D. Zhang, "Mining succinct and high-coverage api usage patterns from source code," in *Proc. MSR*, 2013, pp. 319–328.

[54] M. Acharya, T. Xie, J. Pei, and J. Xu, "Mining api patterns as partial orders from source code: from usage scenarios to specifications," in *Proc. FSE*, 2007, pp. 25–34.

[55] E. Moritz, M. Linares-Vásquez, D. Poshyvanyk, M. Grechanik, C. McMillan, and M. Gethers, "Export: Detecting and visualizing api usages in large source code repositories," in *Proc. ASE*, 2013, pp. 646–651.

[56] H. V. Pham, P. M. Vu, T. T. Nguyen *et al.*, "Learning api usages from bytecode: a statistical approach," in *Proc. ICSE*, 2016, pp. 416–427.

[57] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: finding relevant functions and their usage," in *Proc. ICSE*, 2011, pp. 111–120.

[58] W.-K. Chan, H. Cheng, and D. Lo, "Searching connected api subgraph via text phrases," in *Proc. FSE*, 2012, p. 10.

[59] F. Thung, S. Wang, D. Lo, and J. Lawall, "Automatic recommendation of api methods from feature requests," in *Proc. ASE*, 2013, pp. 290–300.

[60] N. Sahavechaphan and K. Claypool, "Xsnippet: mining for sample code," *ACM Sigplan Notices*, vol. 41, no. 10, pp. 413–430, 2006.

[61] H. Jiang, L. Nie, Z. Sun, Z. Ren, W. Kong, T. Zhang, and X. Luo, "Rosf: Leveraging information retrieval and supervised learning for recommending code snippets," *IEEE Trans. Serv. Comput.*, vol. PP, no. 99, pp. 1–1, 2016.

[62] A. Bacchelli, L. Ponzanelli, and M. Lanza, "Harnessing stack overflow for the ide," in *Proc. RSSE*, 2012, pp. 26–30.

[63] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "Mining stackoverflow to turn the ide into a self-confident programming prompter," in *Proc. MSR*, 2014, pp. 102–111.

[64] J. San Pedro and A. Karatzoglou, "Question recommendation for collaborative question answering systems with rankslda," in *Proc. RecSys*, 2014, pp. 193–200.

[65] J. Cordeiro, B. Antunes, and P. Gomes, "Context-based recommendation to support problem solving in software development," in *Proc. RSSE*, 2012, pp. 85–89.

[66] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Leveraging crowd knowledge for software comprehension and development," in *Proc. CSMR*, 2013, pp. 57–66.

[67] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web?: Nier track," in *Proc. ICSE*, 2011, pp. 804–807.

[68] M. Linares-Vásquez, G. Bavota, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "How do api changes trigger stack overflow discussions? a study on the android sdk," in *Proc. ICPC*, 2014, pp. 83–94.



**Jing Li** Mr. Jing Li is currently a PhD candidate in Computer Science, at the School of Computer Science and Engineering, Nanyang Technological University (NTU), Singapore. His research aims to understand and predict developers' behavior in software development and maintenance, and in social and information networks.



**Zhenchang Xing** Dr. Zhenchang Xing is now a Senior Lectuere in the Research School of Computer Sciecnce, Australian National University. Dr. Xing obtained his PhD degree from the University of Alberta, Canada in 2008. Dr. Xing's main research area is software engineering.



**Ashad Kabir** Dr. Ashad Kabir is a Lecturer in Computing (Mobile Applications) at Charles Sturt University. Dr. Ashad Kabir's research areas include Software Behaviour Mining, Context-Aware Systems, Social Context, Situation Reasoning, Mobile Applications.