Generating Trace Links from Code to Documentation

Breandan Considine
breandan.considine@mail.mcgill.ca
McGill University

Pierre Orhan
pierre.orhan@mail.mcgill.ca
McGill University

Jin L.C. Guo
jguo@cs.mcgill.ca
McGill University

ABSTRACT

When writing software, developers must often switch contexts to navigate between source code, search results and documentation. In practice, writing a query and combing through the search results requires a nontrivial amount of manual effort. In our work, we collect a dataset of embedded links and their lexical context, then learn to rank a set of documents containing the link text, based on their contextual relevance. We compare our relevance-based ranking against a frequency-based ranking using a standard set of information retrieval benchmarks. Finally, we demonstrate a selection-based search assistant to facilitate efficient document retrieval by annotating screenshots containing code-related text with trace links to relevant documentation. In addition to the link dataset itself, we provide two code artifacts for the purposes of experimental reproducibility and artifact evaluation:

- TraceLink (https://github.com/breandan/tracelink)
- TraceJump (https://github.com/breandan/tracejump)

KEYWORDS

document retrieval, selection-based search, semantic search, content recommendation, software documentation, learning to rank

1 INTRODUCTION

[?] to motivate our work "Enable efficient access to relevant content". Documentation is an indispensable resource for software developers learning to use a new language, framework or library. Developer documentation is often published in the form of web pages, and later indexed by a search engine. Consider the typical workflow of a software developer seeking information about an unfamiliar API: to effectively locate relevant documentation, developers must first copy a specific fragment of text (e.g. a function name, error message, or identifier) from their development environment into a search engine, alongside some contextual information. This query must be descriptive enough to retrieve relevant documents with high probability, whilst omitting extraneous information (e.g. user defined paths or tokens) unlikely to occur outside the scope of the developer's personal environment or project. The construction of search queries to rapidly locate relevant documentation is a skill which some have colloquially termed "Google-fu". While power users may prefer lexical search (e.g. to retrieve a known document), lexical search can sometimes impede knowledge discovery by requiring developers to articulate the context of a given query, which is often hard to describe succinctly or in the appropriate jargon. Our work seeks to facilitate knowledge discovery by enriching lexical queries with contextual information extracted from the development environment and by prioritizing contextually relevant documentation sources among the search results.

Once the developer has constructed a search query, she may need to check several apparently plausible documents to locate the desired information. After opening a given search result, the desired content is often buried in a long document. To locate the desired information, the user must then construct a secondary search (e.g. via ctrl / ⌘ + F) to find a specific keyword or phrase, then traverse a list of matches (e.g. via ↵ / ⇧ + ↵), visually scanning the surroundings for relevant information. Lexical occurrences from the list could be sorted based on contextually relevance, instead of the sequential order which they appear in most web browsers.

Prior work in information retrieval for software development has investigated recommending documentation [11] and Stack Overflow content [13] to developers. Our work roughly falls in the same category as Robillard, Treude, et al. Unlike their approach, we do not require manual content curation and support a broad range of languages and documentation types. Rahman et al. [10] explored recommending documentation using lexical similarity, but focuses on a single IDE. More broadly, Dalton et al. [1, 2], El-Beltagy et al. [4] explore entity linking using contextual information in the context of online knowledge bases, but use an outdated bag-of-words (BOW) or n-gram model. Like Dalton et al, our approach prioritizes documents based on semantic or contextual relevance to the query context, but using a sequence embedding. While we focus our attention on linking developer documents in source code, our model is capable of linking entities in any desktop environment or screenshot containing documented code entities.

The rest of this paper is organized as follows. First, we present a novel preprocessing pipeline for extracting links from a corpus of developer documents (or more broadly, any web-based HTML document containing a mixture of source code and natural language). We then describe how to train a language model for ranking a set of lexical search results based on their contextual relevance. We evaluate our ranking function across a standard set of information retrieval benchmarks, such as mean average precision (MAP) and mean reciprocal rank (MRR). We propose a selection-based search assistant for trace link annotation in arbitrary screenshots. Finally, we share a few remarks on the results obtained and potential threats to validity our solution might pose.

2 METHODS

2.1 Preprocessing

To construct our dataset, we use the Zeal User Contributed docsets, a set of archives containing web-based documentation for many common software libraries, scraped from their official documentation sites. From each document in every docset, we extract a set of links and their surrounding text, i.e. *concordances*. More specifically, each link consists of five distinct features:

(1) the source document, i.e. uniform resource identifier (URI)
(2) the target document, i.e. uniform resource identifier (URI)
(3) the anchor text, i.e. a single keyword or keyphrase (query)

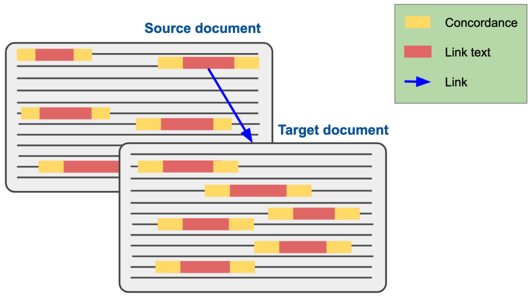


Figure 1: For every link's anchor text, concordances from both the source and target documents are collected.

We then search the entire corpus for the anchor text using either direct-hit or fuzzy matching criteria, and return (4) up to 20 candidate documents mentioning the anchor text most frequently, alongside their title and URI. Finally, we apply a form of query expansion [3] to retrieve (5) every concordance of the anchor text in the target document (see Fig. 1) and the corresponding concordances for each candidate document. [1] During this process, the document text is stored in an inverted index for full-text search (i.e. including terms which may appear outside links in the dataset).

Some links are unrelated to the surrounding concordance. For example, a link may occur inside a table of contents, HTML menu, or boilerplate section. It is further possible, though unlikely, for the anchor text itself to be absent from the target document. To reduce the frequency of atypical links in our dataset, we require all links extracted must conform to the following criteria:

1. Every URI must point to a valid document in the same docset
2. Source and target URIs must not refer to the same document
3. Candidates must contain both source and target documents
4. Anchor text must occur 2+ times in both source and target
5. No link feature may contain any non-ASCII characters
6. Anchor text must fall between 5 and 50 characters in length
7. Anchor text must not contain any whitespace characters

Many of the links collected have natural language concordances which are unlikely to occur within the context of a development environment. Occasionally, links simply do not contain any code tokens whatsoever. We have tried various heuristics to reduce the proportion of such links in our dataset, such as including only links nearby <code> blocks, using orthographic features like naming convention (e.g. camel case, snake case, kebab case) and code-like punctuation (e.g. brackets, colons and other delimiters), but such heuristics are prone to reduce the lexical diversity which legitimate code-like text is likely to exhibit. However inspection of the dataset reveals most links have code-like tokens either in the anchor text or surrounding concordance, as inbound links on API documentation sites often refer to code-related entities. We discuss the implications of the domain mismatch problem further in Section 4.1.

It is possible, although unlikely, for multiple links with the same anchor text and different target URIs to appear within a single document, in which case only the first link occurrence is recorded.

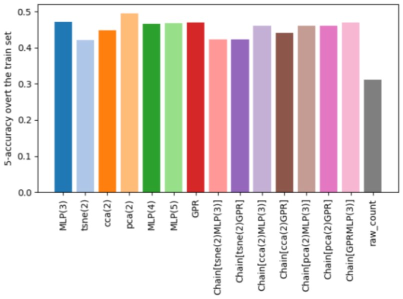


Figure 2: Training set precision using pretrained embedding.

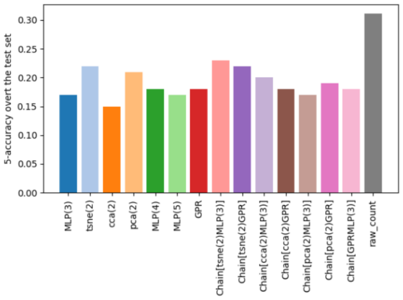


Figure 3: Test set precision using pretrained embedding.

Furthermore, only links with unique features are recorded. For example, if two links with identical concordances are found anywhere in the corpus, only the first occurrence is recorded to prevent imbalance arising from boilerplate text (e.g. which may occur in a table of contents or unrelated page structure).

Our test set is comprised of a random 10% of links from the corpus with exactly the same structure, but with the identity of the target document hidden. Our goal at test time is to predict which document from the list of candidates was the original link target.

3 RESULTS

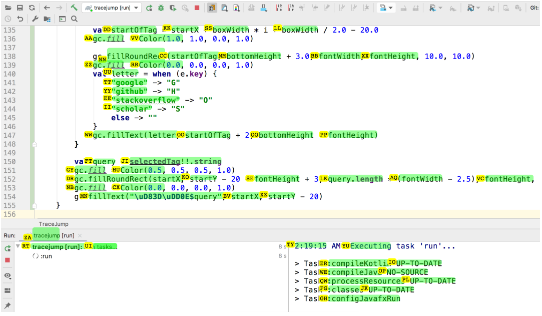
As expected, for link prediction on anchor text contained in the training set, we outperform frequency-based ranking using top-5 prediction accuracy (see Fig. 2), however further training is required for competitive performance on the test set (see Fig. 3).

3.1 Trace Link Annotation

Most trace link annotation tools are based on application-specific workbenches such as DoCIT [5] or embedded as plugins within an integrated development environment such as TraceViz [8]. We present a tool capable of annotating trace links in arbitrary development environments such as terminal emulators, integrated development environments, desktop applications, or even screenshots containing code-related text. Below, we describe the implementation of such a tool, called *TraceJump*.

TraceJump relies on Tesseract [12], an optical character recognition (OCR) library and JavaFX, a library for building desktop applications. To use the tool, the user first invokes a keyboard command (ctrl + \ by default), triggering a screen capture. Using

[1] See: https://gist.github.com/breandan/226bf08bbd5b394a5d2f2a80d383f726



```
            vaDDstartOfTag KKstartX SSboxWidth * i LLboxWidth / 2.0 - 20.0
           AAgc.fill VVColor(1.0, 1.0, 0.0, 1.0)

            gNNfillRoundReCC(startOfTagMMbottomHeight + 3.0BBfontWidthXXfontHeight, 10.0, 10.0)
           ZZgc.fill RRColor(0.0, 0.0, 0.0, 1.0)
            vaUUletter = when (e.key) {
              TT"google" -> "G"
              YY"github" -> "H"
              EE"stackoverflow" -> "O"
              II"scholar" -> "S"
                else -> ""
            }
           WWgc.fillText(letterOOstartOfTag + 2QQbottomHeight PPfontHeight)
        }

        vaFTquery JIselectedTag!!.string
       GYgc.fill HUColor(0.5, 0.5, 0.5, 1.0)
       DRgc.fillRoundRect(startXKOstartY - 20 SEfontHeight + 3LKquery.length AQ(fontWidth - 2.5)VCfontHeight,
       NBgc.fill CXColor(0.0, 0.0, 0.0, 1.0)
        gMNfillText("\uD83D\uDD0E$query"BVstartXXZstartY - 20)
    }
```

```
TY2:19:15 AMYUExecuting task 'run'...

> TaslER:compileKotliIOUP-TO-DATE
> TaslWE:compileJavOPNO-SOURCE
> TaslQW:processResourcePLUP-TO-DATE
> TaslFG:classeJKUP-TO-DATE
> TaslGH:configJavafxRun
```

Figure 4: Trace link annotation with TraceJump.

Tesseract, we perform OCR on the screenshot and filter each term by a TFIDF threshold. Using JavaFX, we highlight a subset of terms exceeding the threshold over the OCR-segmented text (see Fig. 4), which represent expandable trace links. The user can then select a trace link (either directly with their cursor or by typing the adjacent two-letter code), to execute a query over the dataset. Using the inverted index constructed during the preprocessing stage, we retrieve all documents in which the keyword occurs with accompanying concordances. The expanded candidate documents are ranked by their relevance to the screen contents and displayed in KWIC [6] format for document preview, which can be selected to view the corresponding document in a web browser.

4 DISCUSSION

Our results, while preliminary, indicate a relevance-based ranking over search results can surface some documents that a count-based ranking would omit or obscure, using just a pretrained language model for Simple English with some fine tuning. This is encouraging, as it indicates there is untapped potential for using concordance data to train a language model for contextually relevant document retrieval. However further preprocessing and model architecture improvements are likely necessary in order to achieve competitive results on out-of-vocabulary and out-of-distribution queries.

4.1 Threats to Validity

One threat to validity we have identified is related to the problem of domain mismatch. Our application is primarily intended for inferring the relevance of documentation to a source code snippet, based contextual information in the source and target document. Although our corpus consists of inbound links within documentation sites containing frequent references to API entities, the context may be too dissimilar to be informative in a pure code setting. Ideally, the training set would consist of links exclusively containing code tokens drawn from the same vocabulary. Links within documentation pages may be too broad to infer relevance within other content types based on text alone. We believe it is feasible to refine the dataset by using more selective API extraction techniques as described by Ma et al. [7] and Ye et al. [14, 15].

A second threat to validity is the prevalence of autolinking in many knowledge bases and documentation systems. It is common practice in API documentation to use static site generators which are capable of injecting links to named entities in an automatic fashion. We have attempted to mitigate the threat of autolinking by introducing various criteria (cf. Section 2.1) to help preserve the intentionality and relevance of a link in its natural context. While PageRank [9] may have been a stronger baseline in another setting, it assumes documents containing human-generated links, which is not a safe assumption in many software documentation sites.

Partial observability of the TraceJump screen reader is somewhat troublesome, as it can only extract visible text. The application of OCR and screen-reading technology may be less suitable for applications where context is off-screen or distributed unevenly throughout the source document. The OCR tool is prone to misinterpreting certain characters when the font is small or low resolution, and would require further training to be competitive on domain-specific language models. An application-specific integration has the advantage of parsing exact text sequences and would allow the encoder to incorporate context which is not visible on screen. However building application-specific plugins is significantly more time-consuming and less maintainable.

5 CONCLUSION

Searching for documentation is a task that requires devoting some amount of cognitive effort. We present a novel approach to facilitate knowledge discovery by applying language modeling to learn a ranking over concordance-enriched search results from a standard lexical search engine. Our approach works well in the context of lexical search, and can be added to any document retrieval pipeline in a straightforward manner. As a proof of concept, we provide two tools: a visual trace link navigator, capable of annotating links in graphical development environments, and TraceLink, a novel preprocessing architecture for documentation retrieval. Together, these tools cooperate to faciliate knowledge discovery by suggesting relevant documentation to a software development environment. In future work, we hope to evaluate our ranking function on analytics obtained from real users, and introduce a human-in-the-loop reranking functionality.

REFERENCES

[1] Jeffrey Dalton et al. 2013. A neighborhood relevance model for entity linking. In *Proceedings of the 10th Conference on Open Research Areas in Information Retrieval*. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE, 149–156.

[2] Jeffrey Dalton et al. 2014. Entity query feature expansion using knowledge base links. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. ACM, 365–374.

[3] Efthimis N Efthimiadis. 1996. Query Expansion. *Annual review of information science and technology (ARIST)* 31 (1996), 121–87.

[4] Samhaa R El-Beltagy, Wendy Hall, David De Roure, and Leslie Carr. 2001. Linking in context. In *Proceedings of the 12th ACM conference on Hypertext and Hypermedia*. ACM, 151–160.

[5] Jin Guo, Natawut Monaikul, Cody Plepel, and Jane Cleland-Huang. 2014. Towards an intelligent domain-specific traceability solution. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. ACM, 755–766.

[6] Hans Peter Luhn. 1960. Key word-in-context index for technical literature (kwic index). *American Documentation* 11, 4 (1960), 288–295.

[7] Suyu Ma, Zhenchang Xing, Chunyang Chen, Cheng Chen, Lizhen Qu, and Guoqiang Li. 2019. Easy-to-Deploy API Extraction by Multi-Level Feature Embedding and Transfer Learning. *IEEE Transactions on Software Engineering* (2019).

[8] Andrian Marcus, Xinrong Xie, and Denys Poshyvanyk. 2005. When and how to visualize traceability links?. In *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*. ACM, 56–61.

[9] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[10] Mohammad Masudur Rahman, Shamima Yeasmin, and Chanchal K Roy. 2014. Towards a context-aware IDE-based meta search engine for recommendation about programming errors and exceptions. In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE, 194–203.

[11] Martin P Robillard and Yam B Chhetri. 2015. Recommending reference API documentation. *Empirical Software Engineering* 20, 6 (2015), 1558–1586.

[12] Ray Smith. 2007. An overview of the Tesseract OCR engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, Vol. 2. IEEE, 629–633.

[13] Christoph Treude and Martin P Robillard. 2016. Augmenting API documentation with insights from Stack Overflow. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 392–403.

[14] Deheng Ye, Lingfeng Bao, Zhenchang Xing, and Shang-Wei Lin. 2018. APIReal: An API Recognition and Linking Approach for Online Developer Forums. *Empirical Softw. Eng.* 23, 6 (Dec. 2018), 3129–3160. https://doi.org/10.1007/s10664-018-9608-7

[15] Deheng Ye, Zhenchang Xing, Chee Yong Foo, Jing Li, and Nachiket Kapre. 2016. Learning to extract api mentions from informal natural language discussions. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 389–399.