Poster: Which Similarity Metric to Use for Software Documents?

A Study on Information Retrieval based Software Engineering Tasks

Md Masudur Rahman
University of Virginia
Charlottesville, Virginia, USA
masud@virginia.edu

Saikat Chakraborty
University of Virginia
Charlottesville, Virginia, USA
saikat.chakraborty@virginia.edu

Baishakhi Ray
University of Virginia
Charlottesville, Virginia, USA
rayb@virginia.edu

ABSTRACT

Information Retrieval (IR) plays a key role in diverse Software Engineering (SE) tasks. Similarity metric is the core component of any IR techniques whose performance differs for various document types. Different SE tasks operate on different types of documents like bug reports, software descriptions, source code, etc., that often contain non-standard domain-specific vocabulary. Thus, it is important to understand which similarity metrics are suitable for different SE documents.

We analyze the performance of different similarity metrics on various SE documents including a diverse combination of textual (e.g., description, readme), code (e.g., source code, API, import package), and a mixture of text and code (e.g., bug reports) artifacts. We observe that, in general, the context-aware IR models achieve better performance on textual artifacts. In contrast, simple keyword-based bag-of-words models perform better in code artifacts.

ACM Reference format:
Md Masudur Rahman, Saikat Chakraborty, and Baishakhi Ray. 2018. Poster: Which Similarity Metric to Use for Software Documents?. In *Proceedings of 40th International Conference on Software Engineering Companion, Gothenburg, Sweden, May 27-June 3, 2018 (ICSE '18 Companion),* 2 pages.
https://doi.org/10.1145/3183440.3194997

1 MOTIVATION

Measuring document similarity is a key component of any IR technique. A similarity metric measures the similarity between two documents. An IR technique typically computes similarity scores between a query and candidate documents and ranks the latter based on the decreasing value of the similarity score. Thus, it is important to use appropriate similarity metric for different types of SE documents since the notion of similarities may vary for documents containing source code and texts with non-standard English vocabulary.

SE artifacts often contain a diverse set of information including source code, bug reports, project descriptions, API documentation, etc., which can be quite different from natural language [3]. For example, a bug report that primarily contains natural language text with domain specific keywords is linguistically very different than source code or execution traces. In addition, the notion of similarities may also vary for source code, text using domain-specific non-standard vocabulary, and regular text written in natural language. Thus, it is important to use appropriate similarity metric based on their document types. Despite the importance of choosing proper similarity metrics, it is not clear that how suitable standard similarity metrics are for such diverse SE corpora.

In this work, we address this issue by systematically exploring the effect similarity metrics choice for different software artifacts. In particular, we evaluate the effectiveness of four popular metrics: Vector Space Model (VSM) [14], Latent Semantic Indexing (LSI) [7], BM25 [12], and embedding based Word Mover's Distance (WMD) [5] on different SE documents. Each of these metrics has its own benefit and therefore has its preferred domain. For example, in SE literature, VSM [16, 19] and LSI [9, 10] are commonly used, while BM25 is popular in general purpose search engine, and WMD is the state-of-the-art similarity metric for natural language document classification. Recently, researchers start proposing the word embedding based models to improve different SE tasks [15, 18]. Thus, we choose different metrics that are known to be effective for different SE tasks and analyze them thoroughly for SE artifacts.

2 EXPERIMENT

We conduct the experiment on three artifact types and measure the performance of four similarity metrics.

2.1 SE Document Artifacts

We collect document artifacts from GitHub projects (i.e. Textual and Code) and bug-reports [16] (e.g., mixture of code and text) dataset.
Textual Artifacts. We consider GitHub projects' *description* and *readme* content as textual artifacts. Project description on GitHub is often short and concisely represent the project task. On the other hand, the README file of a project that usually contains a detailed description including how to install and run it.
Code Artifacts. We consider GitHub projects' source code information as code artifacts. In particular, we extract *Method & Class name*, *Import Package name*, *API name* from GitHub projects. We use Eclipse JDT [4] framework to collect these information.
Mixture of Text and Code. We use a benchmark bug report dataset (i.e. JDT project) [16] which contains textual descriptions of bugs, execution traces which is mostly code, and source code information where the bug should be located.

2.2 Similarity Metrics

VSM represents documents as N-dimensional vectors where each dimension corresponds to a separate word or term. Then the similarity between two documents is computed as the cosine angle between corresponding vectors.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194997

BM25 is a probabilistic retrieval metric that ranks documents based on the number of query terms present in each document. It treats a particular matching term's importance in the document and query differently and uses document length normalization.
LSI projects a higher dimensional document-term co-occurrence frequency matrix into a lower dimensional latent space to create document vectors. After inferring the lower dimensional vector of both query and documents, cosine similarity can be used to compute the similarity between two vectors.
WMD is based on the concept that similar words should have similar context words [2] thus similar embedding. WMD [5] leverages word embedding [11] to compute similarity between query and documents.

2.3 Experimental Setup

We use similar GitHub projects to collect the ground-truth for the artifacts extracted from GitHub (i.e. description, readme, method & class, import package, and API). If two GitHub projects implement similar functionality (e.g., media player, text editor, etc.) we consider them as similar. Thus if two documents, D_1 and D_2 of a feature (e.g., *description*) come from similar functional project we consider these two as relevant documents. If an IR model (e.g., BM25) can retrieve D_2 when we search with the query D_1, we consider that as a *hit* (i.e. correct retrieval) otherwise a *miss* (i.e. incorrect retrieval). We manually annotate 1590 GitHub projects to its functional categories.

On the other hand, in the case of searching with the bug report, if an IR model can retrieve the source file where the bug is located we consider that as *hit* and otherwise a *miss* [16].

In all cases, we tune the models for each artifact to its optimal performance and use standard mean average precision (MAP) and mean reciprocal rank (MRR) [8] as our evaluation metrics.

2.4 Results

For textual artifacts, we find context-aware models such as LSI and WMD are in general better, while the keyword based bag-of-words (BOW) model VSM performs best for code only artifacts. In contrast, for mixture (i.e. bug reports) documents, BM25 performs the best. Surprisingly, BM25 is not that effective for text only and code only artifacts.

3 IMPLICATION AND FUTURE WORK

Typically, SE researchers use well-established IR techniques to measure document similarity. The advantage of using such techniques is that they are already stable, fine-tuned, and well explored. However, these models are primarily refined for natural language text corpora. From our results, we find that these retrieval models are highly sensitive to the nature of documents. Thus, for a particular IR-based SE task (e.g., bug localization [16]), it is important to understand what type of similarity metric is suitable for its artifact type.

To incorporate special characteristics (i.e. structure and vocabulary) of SE documents, researchers have started adapting different techniques: query reformulations (e.g., [1]), incorporating code structure in IR systems (e.g., [13]), using domain-specific metadata (e.g., [16]), feature extraction (e.g., [17]), and deep learning (e.g., [6]). In addition to these approaches, an informed similarity model choice might lead to an improved performance. In future, we want to investigate how we can incorporate this knowledge into any IR-based SE task. Furthermore, we want to explore whether this informed choice of similarity metrics can improve the overall performance of SE tasks (e.g., bug localization [16]).

ACKNOWLEDGEMENTS

This work is sponsored by the National Science Foundation (NSF) grant CCF-16-19123 and CNS-16-18771. The conclusions of the paper are of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF.

REFERENCES

[1] Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia, and Tim Menzies. 2013. Automatic query reformulations for text retrieval in software engineering. In *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 842–851.
[2] Zellig S Harris. 1954. Distributional structure. *Word* 10, 2-3 (1954), 146–162.
[3] Vincent J Hellendoorn and Premkumar Devanbu. 2017. Are deep neural networks the best choice for modeling source code?. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 763–773.
[4] Eclipse JDT. [n. d.]. "https://www.eclipse.org/jdt/". ([n. d.]).
[5] Matt J Kusner, Yu Sun, Nicholas I Kolkin, and Kilian Q Weinberger. 2015. From word embeddings to document distances. In *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*. 957–966.
[6] An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N Nguyen. 2017. Bug localization with combination of deep learning and information retrieval. In *Proceedings of the 25th International Conference on Program Comprehension*. IEEE Press, 218–229.
[7] Thomas K Landauer and Susan T Dumais. 1997. A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review* 104, 2 (1997), 211.
[8] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.
[9] Andrian Marcus, Andrey Sergeyev, Vaclav Rajlich, and Jonathan I Maletic. 2004. An information retrieval approach to concept location in source code. In *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*. IEEE, 214–223.
[10] Collin McMillan, Mark Grechanik, and Denys Poshyvanyk. 2012. Detecting similar software applications. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 364–374.
[11] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.
[12] Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. Springer-Verlag New York, Inc., 232–241.
[13] Ripon K Saha, Matthew Lease, Sarfraz Khurshid, and Dewayne E Perry. 2013. Improving bug localization using structured information retrieval. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE, 345–355.
[14] Gerard Salton, Anita Wong, and Chung-Shu Yang. 1975. A vector space model for automatic indexing. *Commun. ACM* 18, 11 (1975), 613–620.
[15] Bowen Xu, Deheng Ye, Zhenchang Xing, Xin Xia, Guibin Chen, and Shanping Li. 2016. Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 51–62.
[16] Xin Ye, Razvan Bunescu, and Chang Liu. 2014. Learning to rank relevant files for bug reports using domain knowledge. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 689–699.
[17] Xin Ye, Razvan Bunescu, and Chang Liu. 2016. Mapping bug reports to relevant files: A ranking model, a fine-grained benchmark, and feature evaluation. *IEEE Transactions on Software Engineering* 42, 4 (2016), 379–402.
[18] Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu, and Chang Liu. 2016. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 404–415.
[19] Jian Zhou, Hongyu Zhang, and David Lo. 2012. Where should the bugs be fixed?-more accurate information retrieval-based bug localization based on bug reports. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 14–24.