Evaluating the Specificity of Text Retrieval Queries to Support Software Engineering Tasks

Sonia Haiduc[1], Gabriele Bavota[2], Rocco Oliveto[3], Andrian Marcus[1], Andrea De Lucia[2]
[1]*Computer Science Department, Wayne State University, Detroit, MI 48202, USA*
[2]*School of Science, University of Salerno, 84084 Fisciano (SA), Italy*
[3]*STAT Department, University of Molise, 86090 Pesche (IS), Italy*
sonja@wayne.edu, gbavota@unisa.it, rocco.oliveto@unimol.it, amarcus@wayne.edu, adelucia@unisa.it

***Abstract*—Text retrieval approaches have been used to address many software engineering tasks. In most cases, their use involves issuing a textual query to retrieve a set of relevant software artifacts from the system. The performance of all these approaches depends on the quality of the given query (i.e., its ability to describe the information need in such a way that the relevant software artifacts are retrieved during the search). Currently, the only way to tell that a query failed to lead to the expected software artifacts is by investing time and effort in analyzing the search results. In addition, it is often very difficult to ascertain what part of the query leads to poor results. We propose a novel pre-retrieval metric, which reflects the quality of a query by measuring the specificity of its terms. We exemplify the use of the new specificity metric on the task of concept location in source code. A preliminary empirical study shows that our metric is a good effort predictor for text retrieval-based concept location, outperforming existing techniques from the field of natural language document retrieval.**

***Keywords*-Text retrieval; Query specificity; Concept location.**

I. Problem description

Many software artifacts created during software evolution are written mostly in natural language text (e.g., requirements, design documents, user manuals, scenarios, bug reports, developers' messages, identifiers and comments in source code). The information embedded in these artifacts plays an important role in understanding the software system, as it encodes to a large degree the domain of the software, the developers' knowledge about the system, design decisions, etc.

In order to leverage the information captured in these artifacts, researchers have made use of text retrieval (TR) techniques to support many software engineering (SE) tasks, such as: requirements traceability, refactoring, concept location, impact analysis, defect prediction, bug triage, reverse engineering. The most common way to address these SE tasks with the use of TR techniques is to rephrase the SE task as an Information Retrieval (IR) problem. All such cases rely on the formulation of a textual query by a human or extracted automatically from an existing textual artifact. Many search engines for software artifacts are built around TR methods. Like any search engine, the performance of TR techniques is strongly dependent on the query. When the query is well formulated (i.e., it describes the information need following a discourse similar to the one in the search space), TR works well. However, when the query is ambiguous or too general, TR performs poorly and the queries require rewriting.

Writing a good query is not an easy task for several reasons. First, developers may not be familiar with the vocabulary used to describe particular concepts in the search space (i.e., the collection of software artifacts). For example, when a developer wants to search the source code for the implementation of a feature, she may be unfamiliar with the identifiers used in the source code to refer to particular aspects of the feature. Other factors inherently affect TR techniques in other fields as well. For example, synonyms and homonyms contribute to the ambiguity of a query. Also, TR techniques do not use predefined vocabularies or grammars, which makes them fast and robust, but also limits their retrieval performance in such cases.

All TR techniques rely on computing rather complex textual similarity measures between the queries and the software artifacts. These are most often opaque to users and in consequence it is hard for them to decide, even after looking at the results of a query, which terms in the query are good and which ones are not. In consequence, the effort spent on investigating irrelevant results is often daunting. This problem is not specific to software; it was first encountered in the field of natural language document retrieval [1]. Researchers in this field have addressed the problem by proposing several techniques to measure various aspects of the quality of queries: specificity, coherence, conciseness, etc. These metrics can help users determine which parts of the queries should be rewritten.

A. Solution and Motivation

Our long-term goal is to assist the TR query formulation and selection process for SE tasks. One important step in this direction is estimating the quality of a query (i.e., determining when a query would need reformulation). For this, we draw inspiration from solutions proposed in the natural language document retrieval field. However, TR in SE is not quite the same as TR in natural language corpora,

especially when we are dealing with source code. The rules of discourse and information needs of the users are different in SE that in regular document retrieval tasks. Among the existing aspects of query quality, *specificity* is somewhat independent from the rules of discourse and hence potentially most suited to be used with software artifacts. Specificity measures how discriminative are the terms in the query for describing the current information need. There are two categories of measures: pre- and post-retrieval, based on whether they are computed before or after the query is run. We focused on pre-retrieval metrics as our goal is to offer feedback to the developer about the query as soon as possible. Among these measures, *average inverse document frequency*, or *avgIDF* [2] performs the best on natural language corpora, and it was found to have a relatively high correlation with the retrieval performance. In order to assess if *avgIDF* could be used o predict the quality of the retrieval also on software data, we measured the correlation between *avgIDF* and the retrieval effort on existing concept location data. The weak correlation (see Section IV) indicates that the metric does not work well on software data.

In consequence, we propose a new metric, called *Query Specificity Index (QSI)*, to automatically detect the specificity of queries for TR approaches in the context of SE tasks. The metric is able to measure the expected specificity of a query prior to the retrieval and relies on information theory in order to determine the ability of a query to discriminate between relevant and irrelevant artifacts. QSI can be used in several ways in SE tasks. For example, when dealing with user formulated queries, QSI can be used to recommend the developer to reformulate the query before spending time on investigating the retrieval results. Some SE tasks rely on queries extracted from existing artifacts, such as, traceability recovery. During this process a lot of effort is spent on the manual validation of the retrieved links and on providing feedback to the retrieval system. QSI can be used to prioritize the links that should be investigated first by the users.

In this paper we exemplify the use of QSI in the context of concept location in source code and compare its performance with *avgIDF*. Concept location is the process of identifying where a code change is to start, in response to a change request and many concept location techniques use TR as the underlying mechanism for tool support. We present a case study where QSI was used to estimate the specificity of nearly one hundred queries used in concept location. High correlation between QSI and the retrieval effort indicate that QSI can be used for predicting the effort spent on concept location and that it performs significantly better on software data than *avgIDF*.

B. Related Work

To the best of our knowledge, no previous work addressed the problem of measuring the query quality (in particular specificity) in the context of SE tasks. Within SE, the work most related to our approach deals with the manual or automatic query reformulation and refinement [3], [4], [5], [6], [7], [8], mostly based on relevance feedback mechanisms. These approaches rely on repeated execution of queries and analysis of the results at every run to provide feedback. In contrast, our approach measures the specificity of the query before its execution.

Regarding concept location, Marcus *et al.* [9] proposed TR to support such a task. Many improvements have been developed since then, as reported in a recent survey [10]. All these approaches share a common canonical process: (i) the source code is converted to a text corpus; (ii) the corpus is indexed with a TR technique; (iii) the user writes a text query based on a change request; (iv) the TR engine retrieves a ranked list of source code documents as results; (v) the user inspects the results and, if needed, reformulates the query and returns to step (iv). Our work focuses on step (iii) of this process.

Several studies have investigated the results of formulating different queries for the same information need [11], [12], which highlighted the strong dependence of the TR performance on the query and motivate our work.

II. The Query Specificity Index

We introduce a novel metric to determine the specificity of queries before retrieval in the context of TR-based solutions to SE tasks. The new metric, called *Query Specificity Index* (QSI), is based on concepts from information theory. More specifically, it uses the concept of *information entropy* [13] to measure the specificity of a term in the query. Information entropy measures the amount of uncertainty of a discrete random variable [13]. In our case, the random variable is represented by a term in the query, while the documents in the corpus (i.e., software artifacts in our context) are the possible states that the variable can assume (i.e., the term does or does not occur in an artifact). This means that the more scattered the term is in the corpus the higher its entropy will be. Our conjecture is that a specific query should contain terms that are not very scattered through the corpus, but that are found in a high concentration in few documents (i.e., the relevant ones). We call such terms *specific terms*. While *avgIDF* is also based on the principle that terms with a low scattering are more specific, it overlooks one important aspect: the concentration (i.e., the frequency of terms in the documents where they appear). In consequence, it does not make a distinction between terms that are found many times in a few documents in the corpus and terms that are found only once in few documents in the corpus. QSI addresses this limitation by considering also the concentration of terms in documents, along with their dispersion.

Formally, the entropy of a term t is computed as $entropy_t = \sum_{d \in D_t} p(d) \cdot log_\mu p(d)$, where D_t is the set of documents containing the term t, μ is the number of

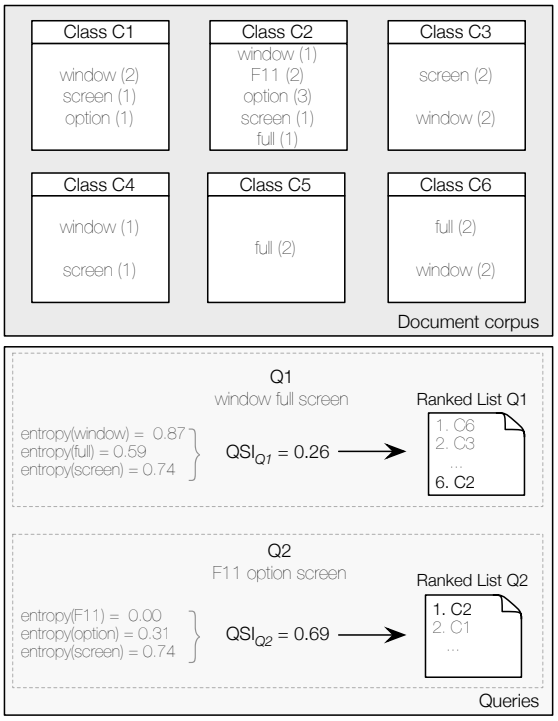

Figure 1. QSI for two different queries.

documents in the corpus, and $p(d)$ represents the probability that the random variable (term) t is in the state (document) d. Such a probability is computed as the ratio between the number of occurrences of the term t in the document d over the total number of occurrences of the term t in all the documents in the corpus. The entropy has a value in the interval of [0, 1]. The higher the value, the less the term is discriminating. We thus compute the QSI based on the entropy of its terms as $QSI_q = 1 - median\{entropy_t \mid t \in q\}$. We chose to use the median over the average because the median is less impacted by skewed distributions of values, caused by a few non-specific terms that may occur in otherwise highly specific queries. Hence, we avoid situations where a few terms have a strong impact on the QSI.

In the rest of the paper we use and discuss QSI in the context of concept location (CL) in source code. Figure 1 shows an example of computing the QSI of two different queries formulated for locating the code related to a change request. The software is composed of six classes (C_1–C_6) and each is converted to a document in the corpus. The change request is actually a bug report: *The window containing the user interface does not scale to full screen when pushing the F11 button on the keyboard*. The goal of CL in this example is to identify the class containing the bug (in this particular example is C_2). Figure 1 shows the two queries (i.e., Q_1 and Q_2) as well as the document corpus. The figure also lists the terms used in the two queries that occur in each of the classes (the number of occurrences appears in parenthesis). As we can see, query Q_1 contains terms having very high entropy. All the terms from Q_1 appear in several classes of the system and thus they do not help much in discriminating between the documents. For this reason the query specificity of Q_1 is not high ($QSI_{Q1} = 0.26$). As we can see, when executing Q_1, the relevant class (C_2) is the sixth class in the ranked list. Thus, the developer would need to investigate six classes in order to locate the faulty one. Conversely, the terms in Q_2 are well focused on a particular document in the corpus (C_2), thus exhibit a low entropy and consequently a high QSI ($QSI_{Q2} = 0.69$). With such a query the relevant class is the first class in the ranked list, indicating a minimum effort spent on CL. Note that low entropy does not necessarily imply that the documents that the query is focused on are the relevant ones.

Table I
PRELIMINARY EVALUATION: LINEAR CORRELATION ANALYSIS

System	Version	#Methods	#Queries	Correlation	
				avg_{IDF}	QSI
Adempiere	3.1.0	28,354	34	-0.17	**-0.72**
ATunes	1.10.0	3,480	30	**-0.52**	-0.43
JEdit	4.2	5,532	30	-0.16	**-0.47**
Overall	-	37,366	94	-0.07	**-0.53**

III. PRELIMINARY EVALUATION

Our goal is to assess if query specificity metrics can be used as indicators of the effort spent on TR - based CL (which we consider as a measure of retrieval quality). We use a metric that approximates this effort as the number of retrieved results (i.e., source code documents) that a developer needs to examine before finding the first relevant document to the change (we assume the developer is examining the results in the order provided by the TR engine). This effort measure is commonly used in the existing research that empirically evaluates CL techniques [10]. We measure the correlation between the $avgIDF$ and QSI, on one hand, and the CL effort measure, on the other hand.

We use change data from three open source systems, namely Adempiere[1] 3.1.0, ATunes[2] 1.10.0, and JEdit[3] 4.2. For each object system, we selected a set of change requests from its issue tracking system corresponding to bugs present in the investigated version of the software, but fixed in a later version. We determined the set of methods that were modified in order to fix each bug, which we used then as the golden set for CL. We will refer to these methods as the *changed methods*. For each change request, we created two queries, extracted from the online issue tracking systems: the first query was composed from the title of the change request (i.e., short description) and the second query composed from the long description of the change request. Table I reports the number of queries used for each system.

[1]http://www.adempiere.com/
[2]http://www.atunes.org/
[3]http://jedit.org/

We used TR in a rather standard way in this study. We built the source code document corpus considering every method in the system as a separate document. For each method we extracted the text found in its identifiers and comments in the source code. We then normalized the text using identifier splitting, stop words removal (i.e., we removed common English words and programming keywords) and stemming. The same normalization techniques were applied on the extracted queries. The corpus was indexed by with Lucene[4], a popular implementation of the Vector Space Model.

The CL effort for a given query is the highest rank of any of the changed methods in the ranked list of search results. As mentioned before, this is a standard measure used when evaluating CL techniques [10]. We computed the Pearson product-Moment Correlation Coefficient (PMCC) [14] between the values obtained for $avgIDF$ and QSI, respectively, and the CL effort measure for each of the queries. PMCC is a measure of correlation between two variables X and Y defined in $[-1, 1]$, where 1 represents a perfect positive linear relationship, -1 represents a perfect negative linear relationship, and some value between -1 and 1 indicates the degree of linear dependence between X and Y. Cohen et al. [14] provided a set of guidelines for the interpretation of the correlation coefficient. It is assumed that there is no correlation when $0 \leq \rho < 0.1$, small correlation when $0.1 \leq \rho < 0.3$, medium correlation when $0.3 \leq \rho < 0.5$, and strong correlation when $0.5 \leq \rho \leq 1$. Similar intervals also apply for negative correlations.

Table I reports the correlation values for each of the three object systems. As it can be observed, QSI achieves a higher correlation than avg_{IDF} on two of the three object systems. Only for *ATunes* the correlation obtained by avg_{IDF} is higher (-0.52 vs -0.43). However, when considering the average over the entire set of queries, we can see that QSI has a strong correlation with the CL effort (-0.53), whereas avg_{IDF} has no correlation.

IV. Conclusion and Future Work

We found that $avgIDF$, one of the best query specificity metrics used in natural language document retrieval, does not work very well on software corpora. We conjectured that accounting for term density, in addition to dispersion, we can obtain a better specificity measure for software data. We proposed the QSI, which uses information entropy to counteract the weakness of $avgIDF$. The results of a preliminary evaluation study showed that QSI correlates highly with concept location effort when searching for methods to change in response to bug reports.

The results are promising enough to warrant further work. Our future work focuses on three directions. First, we will investigate combinations of QSI with other query quality metrics with the goal of obtaining better correlations with retrieval quality. Second, we will build recommenders (based on QSI and other metrics), which will suggest users what terms to change in their query, based on their quality, in order to reduce concept location effort. Finally, we will use these query quality metrics to support other SE tasks, such as traceability link recovery.

[4]http://lucene.apache.org/

References

[1] D. Carmel and E. Yom-Tov, *Estimating the Query Difficulty for Information Retrieval*. Morgan and Claypool Publishers, 2010.

[2] S. Cronen-Townsend, Y. Zhou, and B. Croft, "Predicting query performance," in *Proc. of SIGIR*, 2002, pp. 299–306.

[3] A. De Lucia, R. Oliveto, and P. Sgueglia, "Incremental approach and user feedbacks: a silver bullet for traceability recovery," in *Proc. of ICSM*, 2006, pp. 299–309.

[4] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker, "A machine learning approach for tracing regulatory codes to product specific requirements," in *Proc. of ICSE*, 2010, pp. 155–164.

[5] G. Gay, S. Haiduc, A. Marcus, and T. Menzies, "On the use of relevance feedback in ir-based concept location," in *Proc. of ICSM*, 2009, pp. 351–360.

[6] M. Gibiec, A. Czauderna, and J. Cleland-Huang, "Towards mining replacement queries for hard-to-retrieve traces," in *Proc. of ASE*, 2010, pp. 245–254.

[7] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods." *IEEE TSE*, vol. 32, no. 1, pp. 4–19, 2006.

[8] M. Petrenko, V. Rajlich, and R. Vanciu, "Partial domain comprehension in software evolution and maintenance," in *Proc. of ICPC*, 2008, pp. 13–22.

[9] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," in *Proc. of WCRE*, 2004, pp. 214–223.

[10] B. Dit, R. M., M. Gethers, and D. Poshyvanyk, "Feature location in source code: A taxonomy and survey," *JSME*, to appear, 2012. [Online] http://www.cs.wm.edu/~denys/pubs/JSME-FL-SurveyCRCV1.pdf

[11] J. Starke, C. Luce, and J. Sillito, "Searching and skimming: An exploratory study," in *Proc. of ICSM*, 2009, pp. 157–166.

[12] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich, "Feature location via information retrieval based filtering of a single scenario execution trace," in *Proc. of ASE*, 2007, pp. 234–243.

[13] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Wiley-Interscience, 1991.

[14] J. Cohen, *Statistical power analysis for the behavioral sciences*, 2nd ed. Lawrence Earlbaum Associates, 1988.