Large-scale Heterogeneous Program Retrieval through Frequent Pattern Discovery and Feature Correlation Analysis

Bo Liu¹, Liang Wu², Qiuxiang Dong³, Yuanchun Zhou²
¹NEC Labs China, Beijing, China
²Computer Network Information Center, Chinese Academy of Sciences, Beijing, China
³Institute of Software, Peking University, Beijing, China
liu_bo@nec.cn, wuliang@cnic.cn, dongqx@infosec.pku.edu.cn, zyc@cnic.cn

Abstract—In the era of big data, information retrieval becomes even more challenging since the size of data volume is emerging fast and it is difficult to find the right information from the huge amount of heterogeneous datasets. Especially in software engineering domain, it tends to be more difficult to retrieve the right program from projects that are written in different languages and not well-developed. Prior work solved this problem by extracting words from programs, which cannot fully exploit the information of source code. In this paper, we propose a novel program retrieval method by extracting the frequent patterns and analyzing their correlations with accompanying text information. The experimental results on large-scale and heterogeneous datasets validate the effectiveness of our proposed approach. The inferred semantics of programs can significantly improve the accuracy of code artifact retrieval.

Keywords—Information retrieval; big data; data mining; semantics

I. INTRODUCTION

Due to the upcoming data deluge, data analytics and knowledge management have presented significant challenges in various domains [1,2]. As the data volume increases and diverse data sources mix together, how to search the right information from a huge amount of heterogeneous datasets tends to be more and more difficult. Especially in software engineering domain, the retrieval of programs often has low precision since many projects are not well developed, written in different languages and lacking sufficient comments and common semantic understanding.

Traditionally, Antoniollet al. [3] utilizes the words appeared in programs to capture the meanings of a snippet of programs. Latent Semantic Indexing [4] and Language Modeling [5] have also been used to compress representations of programs. The performance of these methods is not satisfactory enough because of limited meaningful words and distinct representations in programs.

To address these problems, this work for the first time proposes a program retrieval method that leverages the functional information of source code for improving the precision of code retrieval to our best knowledge. First the frequently occurred code patterns from programs are extracted, which may be useful for representing programs. These frequent patterns, often ignored by most code retrieval researchers, perform specific actions and are meaningful to understand the code snippets. Second, since some codes and texts are correlated, e.g., descriptions, comments and specifications can be treated as links between codes and texts, we leverage a feature correlation analysis method to infer the semantics of the code snippets.

II. FEATURE EXTRACTION

Most existing works on source code retrieval [3, 6] regard code programs as general documents and build text files by picking up words from comments, identifier types, and function names of codes. They only consider the text information within a code, not exploiting the correlation between codes and texts, thus resulting in low retrieval accuracy. In this work, we propose a method for feature extraction of source codes and the computation of heterogeneous similarity between codes and texts.

Taking Java language as an example, we first extract the class references of Java as the code features. Examples of Java class references are given in Figure 1. This information is normally announced in the top lines of a code file and reveals the relationships between different class files.

It is obviously unwise to choose all these references as features, since there may exist noises in them which are not informative for the retrieval task. Thus, we only select the closed frequent patterns [7] of class references as features. The selected features are directly derived from codes and are connected to word features through correlation analysis.

![Figure 1. Class reference of a Java class.](image-url)

III. FEATURE CORRELATION ANALYSIS

Since the similarity between codes and texts cannot be directly computed, we adopt the Cross modal Factor Analysis (CFA) model [8] to measure the similarity between code features and text features. The model is to produce a transformation matrix for each media, and project the heterogeneous media to a common space. Then the similarity can be measured in the new space.

Two data matrices are first generated: $X$ is the “code feature - code file” matrix and $Y$ is the “text feature - code file” matrix. In order to link them, two transformation matrices are introduced: $P$ and $Q$. Thus, our objective is to
minimize the function: \[ \|PF^T X - QF^T \|^2. \] The orthogonal constraint is that: \[ PQ^T = Q^2, \] which is to reduce the information redundancy among different dimensions of \( P \) and \( Q \), as well as control the scale of \( W \) to avoid overfitting. Such \( P \) and \( Q \) can be generated through singular value decomposition: \[ PQ = U \Sigma U^T. \]

Based on the two transformation matrices, we use the feature correlation matrix \( U_{cor} = PQ^T \) where \( m \) is the number of text features and \( n \) is the number of code features, to link the two media. When we have a data matrix of queries \( Q_{cor} \) and a data matrix of code files \( C_{cor} \), where \( a \) is the number of queries and \( b \) is the number of code files, the similarity matrix between queries and code files can be calculated as \( S_{cor} = QUC^T \).

Figure 1 illustrates the proposed method of our work. We transform code vectors by a “code feature - word” transformation matrix. After that, codes and texts share a common format and thus their similarity can be measured. By integrating text-code similarity and text-text similarity, the retrieval accuracy can be further improved.

IV. EXPERIMENTS

To build up the heterogeneous code retrieval system requires two phases: 1) extracting features from source codes collected from different projects; 2) learning transformation matrices for the projects. Since the feature extraction process is project independent, we deploy a Hadoop system for feature selection in parallel, which largely relieves the time expenses of training. Matrix calculation is accelerated by a distributed algorithm based on MPI (Message Passing Interface). In order to make the results reproducible, code archives are obtained from real-world open-source projects: Eclipse (https://www.eclipse.org/), an IDE developed in Java; and Filezilla (https://filezilla-project.org/), a cross-platform FTP software developed in C/C++. Their code files are used for the retrieval test and the titles of their bug reports and change logs are crawled as queries. The bugs that contain a clear connection to code files are selected in the experiment. The query set contains over 1700 bug reports.

We use twometricsto evaluate the retrieval results: precision at top \( n \) results (P@\( n \)), and recall at top \( n \) results (R@\( n \)).

COSine with Text-Code (COSTC) is our proposed method, which leverages the text-code similarity generated by CFA. Since COSTC is independent of textual methods, it can be incorporated to existing systems. Sowe only adopted COS [9] as the baseline to validate code features and CFA. Tables 1 and 2 present the precision and recall on Eclipse Dataset. Tables 3 and 4 show the results on Filezilla dataset. It can be concluded that COSTC outperforms COS on all metrics in both datasets. The results validate that the text-code similarity is beneficial to the performance improvement.

V. CONCLUSION AND FUTURE WORK

This work proposes a novel program retrieval method by extracting the frequent patterns and analyzing the correlations between texts and codes. Our method significantly outperforms the baseline, which validates the effectiveness of extracting features from codes. In the future, we will explore more descriptive features from source codes, such as the structure and functional modules of codes, and test our model on more complex datasets with different programming languages and projects.

REFERENCES


