Fuzzy Systems and Data Mining VI
A.J. Tallón-Ballesteros (Ed.)

doi:10.3233/FAIA200728

Recommendation Based on Java Code Analysis and Search

Shanqing FU[a], Bing LI[a], Yi CAI[a], Zhuang LIU[b], Junxia GUO[a,1]

[a] *College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China*
[b] *China Beijing Environment Sanitation Engineering Group Co, Beijing 100101, China*

Abstract. How to improve the efficiency and quality of software development is an ongoing concern in the field of software engineering. As a useful auxiliary function, code recommendation is embedded in almost all integrated development environments. There has been increasing interest and research in the area of code recommendation in recent years due to its convenience for project development. Existing research has made a lot of contributions to this field, but there are still many issues that need further study. One of the key points is the low success rate of recommendation. Focusing on this problem, this paper proposes a method to recommend Java source code after parsing massive amounts of source code information. We propose a new source code analysis algorithm for the scraped source code data. A source file is parsed into classes, methods, and attributes as recommendation objects. At the same time, the annotation information is bound to the annotated objects. Finally, the parsed information is indexed at the project, class, and method levels for code recommendations in a hierarchical recommendation manner. A code recommendation system is implemented by combining this with full-text retrieval technology for class library, class, and method level recommendation. The experimental results show that the method proposed in this paper has better performance in recommendation accuracy than existing code recommendation engines.

Keywords. code search, code analysis, static code recommendation

1. Introduction

The Integrated Development Environment (IDE) is an essential tool for programmers, especially when developers are starting to work with a new code base. The main existing main IDEs provide practical code recommendations or completion of auxiliary functions. In actual project development, the code recommendation function provides recommendations such as methods, attributes and parameter lists, which can shorten development time and improve development efficiency. Moreover, research shows that code recommendation is favored by Java developers [1].

[1] Corresponding Author: Junxia GUO, Beijing University of Chemical Technology, Beijing 100029, China. E-mail: gjxia@mail.buct.edu.cn

The work described in this paper is supported by the National Natural Science Foundation of China under Grant No.61702029, No.61872026 and No.61672085

In the 1970s and 1980s, the style of software development was still to write most of the code by oneself. Today, with the rapid development of the Internet, this situation has been greatly changed [2]. Much research has shown that software and code reuse improves software development efficiency [3]. The lack of experience of software developers, or skilled developers, hoping to find the existing software or code to help them save development time means that, a lot of time is spent searching the Internet. Therefore, how to lessen the amount of search to meet their code requirements deserves further study.

Many code search engines (e.g., Krugle [4]) have large common codebases, where programmers can obtain relevant code through searching. However, these engines only provide keyword search functions. Typically, keyword matching does not mean that there are any similarities in software functionality. Lv [5] established an index domain of method entities by extracting code text, descriptive names, API calls, and other features, but this method ignores the correlation between code-related feature information outside the method and method entities, such as comments and class names. HILL [6] considered a variety of code characteristics of the program; however they didn't distinguish between classes and methods in the program comments. Therefore, to consider more code-related information and to make class and method distinctions, this paper considers code recommendation from a new perspective by, selecting projects, classes, and methods as recommended objects, and proposes a new program analysis method and index structure. For different recommended objects (projects, classes, methods), the code characteristics we considered are different, and we establish different indexes according to the parsed relevant information. Also, the annotation information is bound to its object for improving the accuracy of code recommendation.

2. Related Work

2.1. Program Analysis

For the recommendation of Java static language, the recommended class, method and other information are included in the Java source file. This is not structured data, so code analysis should be completed first. Most of the existing code recommendation methods are based on program static analysis [7]. For example, Grechanik [8] analyzed API calls and their description information in the program, and recommended them to programmers by searching for keywords matching the API information. Li et al. [9] used a program static analysis tool to process the source code. This identifies method calls and the variable define use and other dependencies, confirm the annotation passing path, and extracts the text from the annotation before method definitions. These methods provide a large number of references and have their own characteristics and advantages. However, they are not perfect. The problem is that the amount of feature information extracted by the above methods is insufficient, and the correlation between the recommended object and the feature information under a variety of feature information is not considered.

2.2. Code Recommendation Based on Search

The goal of recommendation system is to recommend information or products that users are interested in according to their information needs and interests. At present,

such systems have been widely used in various fields [10][11][12]. Most search-based code recommendation systems are based on information retrieval technology, which can quickly retrieve large-scale codebases. In the field of code search, the engines are mainly based on keywords [13], input and output [14], and interface [15]. Wu et al. [16] found that the methods retrieved by programmers cannot directly meet their needs, and the changes made by programmers to the methods also reflect their needs. Therefore, starting from the possible intention of programmers to alter methods after retrieval, the aim is to predict this possibility, and use it for query expansion, so as to improve the accuracy of recommendation. LV [5] extracted features through a parsing program, established an index field of method entities, and constructed a Boolean logic query expression through query statements and the related API to help improve the accuracy of code search.

3. Code Parsing and Index Creation

This section introduces the processing of massive amounts of source code and the Java code recommendation model. The framework is shown in Figure 1, which mainly includes data crawling, code parsing, and hierarchical recommendation. First, we construct the codebase which contains a lot of source code. This paper constructs a crawler system to provide source code data for the whole recommendation process. Second, we parse the crawled source code and convert it into class, method, attribute, and other recommended objects; then we segment its words and establish a hierarchical index. When a user enters a query statement, the recommendation system will carry out full-text retrieval according to the keywords and filter conditions in the query statement and score the relevance of the retrieval results. Finally, the system recommends the corresponding project name, class, or method that can realize the relevant functions to the user.

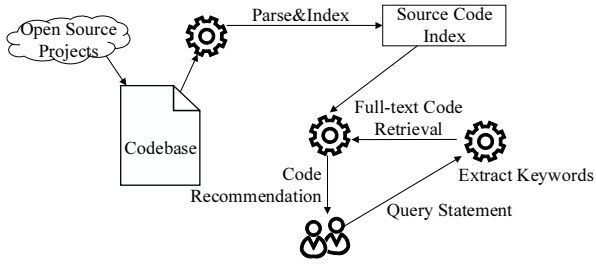

Figure 1. The framework of our code recommendation model

3.1. Code Analysis

Each structure of a Java program has its own rules. For example, a ".java" file can contain the following structures: a package declaration, multiple import declarations, multiple multiline/single-line comments inserted anywhere, multiple annotations using enum types, interfaces, classes, annotations, the definition of multiple enums, annotation structure, etc. Each structure in a Java program has its own characteristics which are different to others, and its own syntax rules. All structures form code blocks independently and do not interfere with other structures.

Based on the above ideas, aiming at the characteristics of Java source code, Algorithm 1 is proposed to analyze the various structures in Java source files mentioned above. Before parsing the code, it is necessary to make clear the information contained in various structures and the criteria for judging each structure. Taking the class structure as an example, each class structure of a Java file can contain information such as static/non-static member variables, method definitions, general comments, Javadoc annotations, usage of annotations, internal classes, interfaces, annotation definitions, enum type definitions, static/non-static code blocks, etc. The criterion for judging the current code as a class is that when a '{' is first detected, the character before it contains the 'class' keyword which is not part of another string.

Based on a clear understanding of the information and criteria that each structure of the Java code file contains, the Java source file is traversed. When a character or string that can indicate a structure (such as enum, class, method, etc.) is identified, it is processed recursively in the processing function of the structure and the information contained in the structure is obtained. When the structure is resolved, the processing continues by tracing back to the structure that recursively calls the method. This is repeated until all the structures of the entire file are parsed. With this algorithm, Java code files can be fully parsed into structured data. Based on this data, full-text retrieval indexes are constructed and data support is provided for implementing hierarchical recommendations.

Algorithm 1 Java program parsing algorithm
Input: Java program file
Output: The feature information of the program

```
1：packInfo=getPackage(JavaFile)
2：impoInfo=getImport(JavaFile)
3：P←JavaFile.readlines()
4：while P != null do
5：If type(P) == enum then
6：enumMembers=enumProcessor(P)
7：else if type(P) == interface then
8：interMembers=interProcessor(P)
9：else if type(P) == class then
10：classMembers=classProcesser(P)
11：else if type(P) == method then
12：methodMembers=methodProcesser(P)
13：else if type(P) == annotation then
14：annotMembers,=annotationProcessor(P)
15：end if
16：P=P.next()
17：end while
18:return(packInfo,impoInfo,enumMembers,iterMembers,classMembers,annotMerbers)
```

The annotation information of the program is very important for code search. When the algorithm recurses downward, it can transfer the annotation information that has been resolved by the parent structure of the current structure, and attach the recently generated annotation information (not attached to other structures) to the structure.

3.2. Index Creation

After parsing the source code, the extracted information becomes a new kind of representation of the code. Due to the different information of projects, classes and methods, in order to achieve hierarchical recommendation, the extracted information is divided into project domains, class domains, and method domains, and we establish the index respectively.

Elasticsearch [17], as a commonly used full-text retrieval tool, and is an open-source real-time search and distribution engine based on Lucene [18]. Based on Elasticsearch, we establish different indexes for different recommended objects (project, classes, methods).

Table 1. The keywords contained in classname and description fields

Class number	Field	Keywords
1	Classname	Set
	Description	collection,contains, duplicate,elements
2	ClassName	List
	Description	Ordered,Collection

Table 2. The classes corresponding to keywords

Field	Keywords	Class number
Classname	Set	1
	List	2
	Collection	1,2
Description	Contains	1
	Duplicate	1
	Elements	1
	Ordered	2

In the Java language, all methods belong to a specific class in the program. For each method there is the corresponding method-name, class, annotation, method body, and other information. In this paper, each project, class, and method are regarded as recommendation objects, denoted as documents (the objects stored in the indexes in the Lucene framework). Each document will be divided into multiple fields according to different code characteristics. Similarly, for projects and classes, different index fields are constructed. We list the index fields of method level, class level and item level respectively at https://github.com/fushanqing/CodeTip/tree/master/Index.

When users search through query statements, our recommendation system gets a series of keywords token. An inverted index is used to find the project, class, or method containing it through keywords. We create an inverted index for each field. As shown in Table 1 and Table 2, the relationship between the classname field and the description field in the two classes is simply represented.

4. Experimentation and Analysis

To evaluate the effectiveness of the proposed method, a complete code recommendation system is designed. The code for our system is released at https://github.com/fushanqing/CodeTip. The recommendation system uses Elastic Search (ES) as a tool for full-text retrieval. ES provides search services for other services in the form of RESTful APIs. Other services can store documents in ES and send the DSL statements to query matching documents.

4.1. Experimental Settings

This paper chooses the Ali Maven Mirror Warehouse [19] source code dataset as the data source with a total size of 1Gb. In the Maven project, users can uniquely identify a

project for a Maven mirror warehouse by groupId, artifactId, and version. Considering that there are a large number of related methods and research in the field of code recommendation, this experimentation uses 32 query statements, which is listed at https://github.com/fushanqing/CodeTip/tree/master/Query, based on previous studies [20][21].

In order to provide a better evaluation of this work, two research questions are raised.

Question 1: What is the accuracy of our method?

To answer this question, 32 queries are used in the experimentation, in which the first 15 query statements are the same with as Krugle [4] searches for comparison.

Question 2: What is the recommendation performance of our method in graded recommendation?

To answer this question, the experiment analyze the query-related recommendations at the higher-ranked project, class, and method levels to determine the accuracy of recommendations at these levels.

4.2. Measurement and Evaluation

In the information retrieval system, there are a series of indicators that can be used to evaluate the questions raised in the previous section comprehensively and accurately. The common indicators are: precision of the first N results (Precision@N), mean reorder reciprocal(MRR), precision-recall, mean average precision (MAP), etc.

We analyzed these evaluation indicators and considered that the objective of this study is to achieve high precision code recommendation, so Precision@N and MRR are selected as the evaluation indicators here.

4.3. Experimental Results

In order to reduce the impact of human subjective judgment on the experimental results, we use the comprehensive judgment of two programmers for the evaluation of the recommended results. Only when both programmers think that the recommended results are related to the query will the recommendations be recorded as correct.

In order to answer Question 1, the first 15 query statements are tested against Krugle to compare the correlation of the recommended method code. As can be seen from Table 3, in the experimental comparison of the first 15 query statements, our method results in an overall improvement over Krugle in sorting the related results, which indicates that our method has a higher recommendation precision. Among the query statements used for the experimentation, some are highly relevant to the recommended results, such as ‘how to split a string into words’, but there are still individual query statements that are not recommended as well, such as ‘how to Deserialize an XML document’. The reason for this is that there is no code base associated with the query statement in the code depository.

A summary of 32 selected query statements is shown in Table 4. It can be seen that each index value decreases with the increase of search sentences, but the above two experimental results show that this method is effective in improving the search accuracy. The experimental results for each query statement are shown at http://github.com/fushanqing/CodeTip/tRee/master/experiment_Results.

Table 3. The comparison of experimental results

Indicators	Our method	Krugle	Improvement
Precision@1	73.3%	70%	3.3%
Precision@5	57.3%	29.4%	27.9%
MRR	81.3%	60%	21.3%

In order to answer Question 2, the experimentation also uses all the query statements to compare the recommendation at project and class level with the recommendation directly. The experimenter can choose a specific project (or class) to judge the correlation between the method code recommended in the project(or class) and the query statement. The experimental results are shown in Table 5.

Table 4. Experimental results for all query statements

Indicators	Our method
Precision@1	68.8%
Precision@5	55%
MRR	77.7%

Table 5. Experimental results at different levels

Indicators	Project	Class	Method
Precision@1	53.3%	60%	73.3%
Precision@5	34.7%	42.7%	57.3%

As can be seen in Table 5, our method can recommend code related to query statements at the project and class levels. The recommended accuracy for a specific class is higher than that for a specific project, and the two levels of recommendation have slightly less precision than that of direct method level recommendation. The above results show that the code recommendations methods at the project, class, and method levels in this paper are valid.

5. Conclusion

In order to address the problem of low accuracy in the current code recommendation field, this paper proposed a Java code recommendation method based on full-text retrieval. After obtaining the Java source code information through the Java source code parsing algorithm designed in this paper, we established a hierarchical index for this information, and recommend projects, classes, and methods to users according to query statements. The experimental results show that our method improves the accuracy of the recommendation results.

References

[1] Murphy GC, Kersten M, Findlater L. How Are Java Software Developers Using the Eclipse IDE?. IEEE Software. 2006, 23(4):76-83.

[2] Grechanik M, Fu C. Xie Q, McMillan C, Poshyvanyk D, Cumby C. A search engine for finding highly relevant applications. In:2010 ACM/IEEE 32nd International Conference on Software Engineering, Cape Town; 2010, p. 475-484 .

[3] Andrew JK, Brad AM, Michael JC, Htet HA. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. IEEE Trans. Softw. Eng, 2006, 32(12):971–987.

[4] Krugler K, Krugle Code Search Architecture. In: Sim S.E., Gallardo-Valencia R.E. (eds) Finding Source Code on the Web for Remix and Reuse. Springer;2013, p. 103-120.

[5] Fei L, Hongyu Z, Jian-guang L, Shaowei W, Dongmei Z, Jianjun Z. CodeHow: effective code search based on API understanding and extended boolean model. In: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE ’15). IEEE Press,2015, p. 260–270.

[6] Emily H, Lori P, Vijay-Shanker K. Improving source code search with natural language phrasal representations of method signatures. In: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'11). IEEE Computer Society, USA; 2011, p. 524–527.
[7] Kirkov R, Agre G. Source code analysis-an overview. Bulgarian Acad Sci. 2010, 10: 60-77.
[8] Grechanik M, Fu C, Xie Q, McMillan C, Poshyvanyk D, Cumby C. A search engine for finding highly relevant applications. In:ACM/IEEE 32nd International Conference on Software Engineering, Cape Town, 2010, p. 475-484.
[9] Li B, Li F, Huang X, He X, Xu L. Transferring Java Comments Based on Program Static Analysis. 2019,638-643.
[10] H. Li, Y. Liu, N. Mamoulis and D. S. Rosenblum. Translation-Based Sequential Recommendation for Complex Users on Sparse Data. In: IEEE Transactions on Knowledge and Data Engineering, 2020,vol. 32, no. 8, pp. 1639-1651.
[11] X. Chen, X. Liu, Z. Huang and H. Sun, RegionKNN: A Scalable Hybrid Collaborative Filtering Algorithm for Personalized Web Service Recommendation.In: 2010 IEEE International Conference on Web Services, Miami, FL, 2010, pp. 9-16.
[12] Wanigasekara N , Liang Y , Goh S T , et al. Learning Multi-Objective Rewards and User Utility Function in Contextual Bandits for Personalized Ranking. In:Twenty-Eighth International Joint Conference on Artificial Intelligence IJCAI-19. 2019.
[13] Fafalios P, Tzitzikas Y. Post-analysis of keyword-based search results using entity mining, linked data, and link analysis at query time. In: IEEE International Conference on Semantic Computing, Newport Beach, CA; 2014, p. 36-43.
[14] Stolee KT, Elbaum S, Dobos D. Solving the Search for Source Code. Acm Transactions on Software Engineering & Methodology. 2014, 23(3):1-45.
[15] Mukund R, Yi W, Youssef H. SWIM: synthesizing what i mean: code search and idiomatic snippet synthesis. In Proceedings of the 38th International Conference on Software Engineering (ICSE '16). Association for Computing Machinery, New York, NY, USA;2016, p. 357–367.
[16] Wu H, Yang Y. Code search based on alteration intent. IEEE Access. 2019, 7: 56796-56802.
[17] https://www.elastich.co/cn/
[18] Gospodnetic O, Hatcher E, Cutting D. Lucene in action. Manning, 2005.
[19] https://maven.aliyun.com/mvn/view
[20] Lemos O A L , Paula A C D , Sajnani H , et al. Can the use of types and query expansion help improve large-scale code search? In: IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM), Bremen; 2015, p. 41-50.
[21] Sushil KB, Joel Ossher, and Cristina V. Lopes. Leveraging usage similarity for effective retrieval of examples in code repositories. In Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering (FSE '10). Association for Computing Machinery, New York, NY, USA; 2010. 157–166.