Stack Overflow Posts as Source of Library Features

Camilo Velázquez-Rodríguez*, Eleni Constantinou† and Coen De Roover‡
Vrije Universiteit Brussel, Belgium, and Eindhoven University of Technology, Netherlands
Email: *cavelazq@vub.be, †e.constantinou@tue.nl, ‡cderoove@vub.be

It is common in contemporary software development to reuse functionality from third-party libraries in order to reduce development time and to improve overall system quality [1]. Each library targets a well-defined domain (e.g., GUI, persistence) and offers functionality to client systems through its API, facilitating the implementation of a particular task within the domain (e.g., displaying a dialog, serialising to JSON). Software ecosystems such as Maven or NPM provide a vast number of libraries that offer different features for reuse.

When selecting a library to reuse from a vast ecosystem, it becomes essential for developers to know the features offered by each library. Kanda et al. [2] consider a set of API methods with a corresponding name as a feature. We generalise their definition in this work so that features comprise the API elements that realise them, and the textual description of the offered functionality.

To alleviate the absence of descriptions of features of libraries, developers often pose questions in Q&A websites like Stack Overflow[1] (SO) to get information about how to realise particular tasks. For example, a post[2] asks about possible ways to delete a directory recursively and a code example of the Apache Commons IO library is offered as a solution. Therefore, the solutions on sites like SO can provide a variety of feature examples that can enhance library documentation, to enable cross-library comparison by users, and to improve recommendation tools of third-party libraries.

We present an approach to extracting the functional features of a library using the library's JAR files and the SO posts where it is used. The latter present a unique opportunity for API mining because of their focus on a particular domain task, and because of the natural language that surrounds code snippets. By verifying the API calls in a snippet w.r.t. the API of the target library, we can ensure the validity of the example.

The approach can form the foundation for exploring an ecosystem through the features offered by its libraries:

1) **Data Collection**: Our approach requires that the *groupID* and *artifactID* of at least one version of a library is in the Maven Central repository. Then, it collects information about its public API from their published JAR files, and example usages of this API from SO snippets. A heuristic based on the tags of the SO answer is used to select code snippets related to the library under analysis. For each selected answer, the following information is collected for subsequent processing: the body of the question, the body of the answer, and the title.
2) **Data Processing**: The approach filters out all answers without a code snippet (e.g., many answers related to the Weka library). We designed a custom-built island grammar based on the official Java Language Specification that is capable of parsing code snippets that are incomplete or syntactically incorrect. A parser based on this island grammar extracts the API information we are interested in, e.g., invocations of both instance and static methods. The approach uses the information extracted from the public API of a library to match types (i.e., classes) and methods that are part of a library.
3) **Data Transformation**: This step is different depending on the type of information to transform, i.e., the text or code from an SO answer. In the case of text attributes such as the bodies of the question and answer and their title, the data is cleaned and then a TF-IDF model is trained. Such a model will allow transforming text into numerical vectors. The distance between the vectors is calculated through the cosine metric resulting in a similarity matrix, e.g., every number d_{ij} in the matrix is the distance between the vectors i and j. Similarly, for code attributes (i.e., the original method names, the method names after splitting camel case, and the API usage itself as extracted by the island parser) a similarity matrix is obtained through the Jaccard index.
4) **Clustering, selecting and naming**: The similarity matrices obtained in a previous step are the input to a hierarchical clustering algorithm. A dynamic tree cutting approach is adopted to select the optimal cutting point in the dendrogram. The ideal cluster would be that one in which the feature is clearly depicted; which might not always be the case. We use the local outlier factor (LOF) to check the most common elements within a cluster. From the textual information, the approach extracts pairs noun-verb or verb-noun. The most frequent pairs (extracted with LOF) will form the cluster name.

At BENEVOL, we intend to present our approach, the results from its evaluation, the limitations we currently have, and the future avenues our work might take.

[1] https://stackoverflow.com
[2] https://stackoverflow.com/questions/779519

References

[1] Samuel A. Ajila and Di Wu. 2007. Empirical study of the effects of open source adoption on software development economics. JSS 80, 9.
[2] Kanda, Tetsuya and Manabe, Yuki and Ishio, Takashi and Matsushita, Makoto and Inoue, Katsuro. 2013. Semi-Automatically Extracting Features from Source Code of Android Applications. TIS E96.D, 12.