Automatic library categorization

Camilo Velázquez-Rodríguez
Vrije Universiteit Brussel
Belgium
cavelazq@vub.ac.be

Coen De Roover
Vrije Universiteit Brussel
Belgium
cderoove@vub.ac.be

ABSTRACT

Software ecosystems contain several types of artefacts such as libraries, documentation and source code files. Recent studies show that the Maven software ecosystem alone already contains over 2.8 million artefacts and over 70,000 libraries. Given the size of the ecosystem, selecting a library represents a challenge to its users.

The MVNRepository website offers a category-based search functionality as a solution. However, not all of the libraries have been categorised, which leads to incomplete search results. This work proposes an approach to the automatic categorisation of libraries through machine learning classifiers trained on class and method names. Our preliminary results show that the approach is accurate, suggesting that large-scale applications may be feasible.

KEYWORDS

Software Ecosystems, API Category, Text Classification

ACM Reference Format:
Camilo Velázquez-Rodríguez and Coen De Roover. 2020. Automatic library categorization. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3387940.3392186

1 INTRODUCTION

Software ecosystems are often defined as a collection of software products developed and in constant evolution in the same environment [9]. Examples of ecosystems that collect software libraries include Maven for JVM-based languages (e.g., Java, Scala, Kotlin, Clojure), NPM for JavaScript, or CRAN for R. We focus on Maven[1] in particular, as the popularity of its supported languages encourages contributions from library developers. Benelallam et al. [1] reported over 2.8 million artefacts in the Maven ecosystem related to more than 70,000 libraries. The disparity is explained by the immutability constraint imposed by the ecosystem, which disallows replacing one version of an artefact by another variant of the same version. As a result, the ecosystem hosts several versions of each library. Soto et al. [12] investigated the diversity in the ecosystem resulting from this constraint in detail. Kula et al. proposed visualisations to study the evolution of systems and their library dependencies [7], as well as library popularity, adoption, and diffusion within an ecosystem [8]. Decan et al. [4] study dependency evolution within additional ecosystems that are supported by a package manager application.

Given the large amount of libraries and library versions within an ecosystem, developers can benefit from assistance in selecting the library that is the most appropriate for a particular task. For Maven, at least two web-based views on the ecosystem have been developed to this end: *SonaType*[2] and *MVNRepository*[3]. The latter supports two types of searches for exploring the vast amount of libraries in its database:

Metadata-based search A developer searches for an artefact providing information such as groupId, artifactId or version.
Category-based search A developer searches for a specific category of artefacts related to her search criteria.

Metadata-based searches are of little help without knowledge of the precise metadata of the artefact. Category-based searches are of help to users who do not have a specific library in mind, but who are exploring the ecosystem or looking for alternatives to an already adopted library.

MVNRepository features 150 library categories covering various domains such as Databases, I/O Utilities or Mocking. The total number of libraries in these 150 categories is 37,934. Considering that the number of libraries reported by [1] is 73,653, over 30,000 libraries in the Maven software ecosystem remain uncategorized.

In this presentation abstract, we report on the first results of our work towards an automated means for assigning libraries to an appropriate category by analysing the byte code of their implementation. Specifically, we consider class and method names as input to five machine learning algorithms. Our hypothesis is that libraries in the same category have similar names for their classes and methods. Hence, a new "uncategorised" library could be automatically classified in one of the existing categories.

2 OVERVIEW OF THE APPROACH

Our approach is based on text classification machine learning algorithms trained and evaluated on a corpus of text extracted from the libraries. We obtain this corpus of text by extracting the identifiers of public classes and methods from a library's JAR file using the Apache BCEL [4] library. For those identifiers using the *CamelCase* naming convention, we separate each of the words.

To feed the extracted corpus of text to a machine learning algorithm, we vectorise it using the popular word embedding approach [10] Fixed-length vectors are generated for each word in our vocabulary (i.e., class and method identifiers extracted through the previous step). The generated vectors capture the context around a

[1]https://repo1.maven.org/maven2/

[2]https://search.maven.org/
[3]https://mvnrepository.com/
[4]https://commons.apache.org/proper/commons-bcel/

ICSEW'20, May 23–29, 2020, Seoul, Republic of Korea
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7963-2/20/05.
https://doi.org/10.1145/3387940.3392186

word, hence it is possible to relate different words by the context around them.

The task of the machine learning algorithm is to learn and predict a discrete label for each vector that corresponds to the *MVN-Repository* category to which the library belongs. We consider five machine learning algorithms to instantiate our approach: Gaussian Naive Bayes (GNB) as well as Bernoulli Naive Bayes (BNB) [6], Support Vector Machines (SVC) [5], *k*-Nearest Neighbors (KNN) [3] and Random Forest (RF) [13].

3 EVALUATION AND VALIDATION

We train and evaluate our approach on five *MVNRepository* categories: *Collections*, *Dependency Injection*, *Http Clients*, *Compression* and *Json* libraries. From each category, we select 15 libraries (i.e., jar files). These 15 libraries are divided into 10 libraries for training and evaluation, and 5 for validation of the approach.

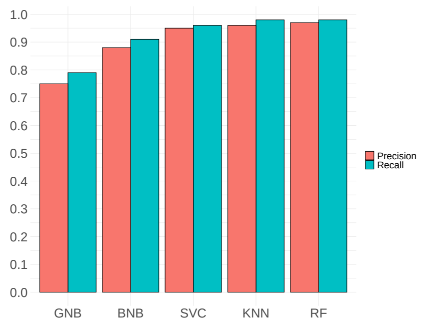

Figure 1: Precision and recall of the approach.

For the evaluation, we use 10-fold Stratified Cross-Validation [11] to train and compute the precision and recall of each classifier model. This technique partitions the data into ten folds of equal size, applying a stratified sampling (i.e., to avoid bias, each fold has the same number of data instances per class to be predicted). At each iteration, a single fold is used as the test set, while the remaining ones are used as the training set. Figure 1 depicts the results. All models produced rather good results, with KNN and RF achieving the highest scores.

The goal of the validation is to further test the scenario in which the model is queried for the category of libraries that were not included in its training phase. Note that this scenario was already covered in the evaluation above, as the model for each of the 10 iterations is trained using 9 training folds and queried using the remaining test fold. In the validation, we stress this scenario further and inspect the results manually. We use the best-performing RF model to predict the category for the 5 unseen libraries in each of the known categories. Out of the five categories that were considered, the predictions for *Collections*, *Http Clients* and *Json* libraries were correct in all cases. For the two remaining categories, the selected model failed to predict the category for one library out of the five considered for each of the categories.

4 DISCUSSION

Like most approaches based on text classification, our approach suffers from *out-of-vocabulary* errors when the model is asked to predict a label for a term it has not encountered in its training phase. These are less likely to arise when the model is trained on sufficiently large data sets. The NLP community has proposed mitigation strategies such as FastText [2], which generates a vector for the unseen term based on the trained representations of its characters. We consider incorporating and selecting the most appropriate mitigation strategy for our problem setting as future work.

Another limitation of our approach is its inability to propose unseen categories. Its training phase relies on the pre-existing categories in *MVNRepository*. Library metadata from other sources (e.g., keywords in repositories) might be required to overcome this limitation. Topic modelling or NLP techniques capable of text summarisation could be explored as an alternative.

5 CONCLUSION

In this presentation abstract, we have proposed a machine learning approach to categorising libraries within software ecosystems. Once trained on a sufficiently large dataset, the approach is capable of assigning an existing library category to a new library. A preliminary evaluation on 5 *MVNRepository* categories of 15 libraries from the Maven ecosystem demonstrates its feasibility. A more extensive evaluation is part of future work.

REFERENCES

[1] Amine Benelallam, Nicolas Harrand, César Soto-Valero, Benoit Baudry, and Olivier Barais. 2019. The Maven Dependency Graph: A Temporal Graph-Based Representation of Maven Central. In *Proc. of the 16th Int. Conf. on Mining Software Repositories (MSR)*.

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.

[3] Belur V Dasarathy. 1991. Nearest neighbor (NN) norms: NN pattern classification techniques. *IEEE Computer Society Tutorial* (1991).

[4] Alexandre Decan, Tom Mens, and Philippe Grosjean. 2019. An Empirical Comparison of Dependency Network Evolution in Seven Software Packaging Ecosystems. *Empirical Software Engineering* 24, 1 (2019), 381–416.

[5] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. 1998. Support Vector Nachines. *Intelligent Systems and their applications* 13, 4 (1998), 18–28.

[6] George H John and Pat Langley. 1995. Estimating continuous distributions in Bayesian classifiers. In *Proc. of the 11th Conf. on Uncertainty in Artificial Intelligence (UAI95)*. 338–345.

[7] Raula Gaikovina Kula, Coen De Roover, Daniel German, Takashi Ishio, and Katsuro Inoue. 2014. Visualizing the Evolution of Systems and their Library Dependencies. In *Proc. of the 2nd Working Conf. on Software Visualization.*

[8] R. G. Kula, C. De Roover, D. M. German, T. Ishio, and K. Inoue. 2018. A Generalized Model for Visualizing Library Popularity, Adoption, and Diffusion within a Software Ecosystem. In *Proc. of the 25th Int. Conf. on Software Analysis, Evolution and Reengineering (SANER18).*

[9] Mircea Lungu. 2008. Towards reverse engineering software ecosystems. In *Proc. of the 24th Int. Conf. on Software Maintenance (ICSM08)*. 428–431.

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NIPS13).*

[11] Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. 2011. On the stratification of multi-label data. In *Joint European Conf. on Machine Learning and Knowledge Discovery in Databases*. Springer, 145–158.

[12] César Soto-Valero, Amine Benelallam, Nicolas Harrand, Olivier Barais, and Benoit Baudry. 2019. The Emergence of Software Diversity in Maven Central. In *Proc. of the 16th Int. Conf. on Mining Software Repositories (MSR19).*

[13] Tin Kam Ho. 1995. Random Decision Forests. In *Proc. of the 3rd Int. Conf. on Document Analysis and Recognition (ICDAR95)*. 278–282.