SN Computer Science (2024) 5:339
https://doi.org/10.1007/s42979-024-02654-2

ORIGINAL RESEARCH

Towards an Automated Classification of Software Libraries

Maximilian Auch[1] · **Maximilian Balluff**[1] · **Peter Mandl**[1] · **Christian Wolff**[2]

Received: 9 February 2022 / Accepted: 24 January 2024

Abstract
Nowadays, the use of third-party libraries in software is common. At the same time, the number of published libraries continues to increase. An automated classification should help to maintain an overview and identify similar software libraries. This paper investigates if new approaches can be used to classify all software libraries crawled from Apache Maven repositories into defined classes using machine learning. In addition to tags that are not always available or of poor quality, we examine one feature that is always available—the id. Consisting of group-id and artifact-id, the id of an Apache Maven software library contains valuable information that can help in classification. Through a developed preprocessing and an optimized recurrent neural network (RNN), the tokenised ids should allow a classification of most libraries. Furthermore, we present an optimized approach through a hybrid use of id tokens and tags in combination. Based on the dataset including 28,600 labeled entries, a comparison of various approaches was carried out. The RNN achieved a balanced accuracy of 71.36% by training on tokenised ids. A model trained on tags achieved a balanced accuracy of 92%. However, the new hybrid approach, which combines tags and ids, optimizes the result to 94.12%. While a classification on tags achieves a better result than the more general id-based approach, the applicability is limited to software libraries that are tagged. The hybrid approach, on the other hand, takes advantage of the classification results based on tags when these are available, but includes valuable information from the always available ids.

Keywords Software libraries · Java · Classification · Machine learning · Recurrent neural network

Introduction

An increasing number of organizations and developers publish software via application programming interfaces (APIs), which contain a range of subroutines and functionalities. These are made available in the form of comprehensive packages, so-called third-party libraries [1]. This allows other software systems to integrate these third-party libraries and gain a usually quick and comparatively easy access to the integrated routines and functionality. Companies with interfaces to digital services can make their products easily accessible and attractive, to offer a service by means of maintaining compatibility and backward compatibility as well as continuous maintenance and further development. The libraries can be used to query services, simplify code, and file processing, integrate prebuild components, simplify tests, analyze code, or integrate complex processing, e.g., in distributed computing. Using libraries, typically less code needs to be written and therefore less code needs to be tested and maintained. As a result, software libraries are an essential part of many software projects in most domains, platforms, and technologies.

The usage of such third-party libraries has increased over the last decades. Back in 2013, Thung et al. [2] analyzed over 1000 projects of considerable size from GitHub. They found that 93.3% of the software projects studied used third-party libraries. On average, these projects each contained 28 third-party libraries. If current uses are analyzed and smaller projects are also considered, this amount may vary considerably. We ourselves examined around 800,000 revisions

This article is part of the topical collection "Data Science, Technology and Applications" guest edited by Slimane Hammoudi and Christoph Quix.

✉ Maximilian Auch
maximilian.auch@hm.edu

[1] University of Applied Sciences Munich, Lothstraße 34, 80335 Munich, Germany

[2] University of Regensburg, Universitätsstraße 31, 93053 Regensburg, Germany

Published online: 27 March 2024

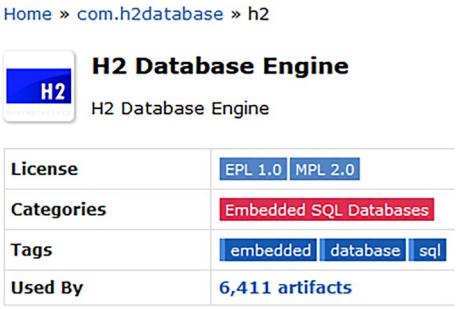

Fig. 1 Example of a categorized and tagged software library on the MvnRepository.com website

of around 15,000 Java projects from GitHub.[1] A random search was performed with the filter criteria that each project should use the dependency management tool Apache Maven and that a Readme.md file should be available in the project directory. First-party libraries were filtered out and only one revision per project with the largest share of third-party libraries was considered. This enabled us to determine an average use of 8.65 third-party libraries per project. The median usage of libraries was 6.

Online available ecosystems, which list or even manage software libraries, are registering an increasing number of projects that can be integrated as software libraries. For example, beside Python Package Index (PyPI) or package management systems such as CRAN (for the R language) or npm (for Node.js packages), Maven is one of these ecosystems. The Maven ecosystem contains libraries that are developed in the context of the Java Virtual Machine (JVM) and made available to it. The indexing platform MvnRepository.com,[2] which is based on different JVM-related repositories like Maven Central[3] for software libraries, provides partly classified and tagged data. The website of MvnRepository.com does not offer an API, but if categories and tags are available for libraries, it does display these as shown in Fig. 1.

A classification of libraries can help developers and other project participants to get an overview of current implementations for wrapped access to technologies, functions, frameworks, and so on. In addition, this can also provide an overview of technologies behind the software libraries within a category. For decision making, it is essential to organize software libraries that are available in that broad spectrum.

Within the context of our overall research project, it is planned to identify a similarity between software projects automatically to derive design decisions and business-relevant information. Software libraries can help to calculate a technical similarity of software by drawing conclusions about similar technical characteristics. Applied software libraries provide information on these characteristics, such as the technology stack in general, types of processing methods and data as well as the type of storage for the data, integration with other systems, security methods, provided APIs, the underlying platform, like application server or cloud-provider, and the usage of the software. For usage, libraries can point to what type of user interface is provided, such as command-line, fat-client, thin-client, and what design pattern is used. By analyzing these characteristics, similar software should be found on a technical basis. In a later step, the evolution of the software over time is to be determined, such as migrations made, and therefore automated decisions about selected technologies, design and architecture are to be derived. However, the problem arises that libraries must be classified to detect migrations from library A to library B with similar functions. As too many libraries are available, manual classification is not feasible. Therefore, this research project aims to evaluate whether libraries crawled from Apache Maven repositories can be classified automatically into defined classes using machine learning (ML).

In the following Section "Related Work", related work is provided, before the new approaches are elaborated in Section "Software Library Classification Approach". Section "Dataset" describes the data required for the training and evaluation approaches, as well as the preprocessing. Section "Evaluation" presents the test setup that is used for the evaluation and introduces the metrics applied for the description of results in Section" Results". Section "Discussion" includes a discussion of the results from the previous chapter. Finally, Section "Threads to Validity" lists some limitations and validity threats before drawing the conclusion in the final Section "Conclusion".

Related Work

Similar to the ongoing growth of repositories for software libraries, app stores[4] have also grown in size over the last decade. As a result, these app stores have already become the focus of research in the past. Various approaches [3] have been used to process and order the apps by means of automated classification or clustering. Yu et al. [4], for example, use not only descriptions but also third-party libraries of mobile apps to perform categorisation and recommendation using similarity metrics and collaborative filtering.

[1] https://github.com/.

[2] https://mvnrepository.com/, last accessed 2021-12-05.

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

[4] The Apple app store (https://www.apple.com/de/app-store/) and Google play (https://play.google.com/store) are frequently used for mobile apps in the studies.

Table 1 Summary overview of the existing published work

Authors	Target	Data-Source	ML-Method
Auch et al. [6]	Multi-class classification of software libraries	Tags	Feedforward neural network, naïve Bayes, random forest, logistic regression
Escobar- Avila [5]	Single-label and multi-label categorization of software libraries	Bytecode, software library profiles and categories	Clustering via vector space model and Dirichlet process clustering
Velázquez- Rodríguez and De Roover [7]	Multi-label classification of software libraries	Tags and word vectors from binary files	Several specific classifier out of a MEKA toolkit
Yu et al. [4]	Categorization and recommendation of mobile apps	Descriptions, third-party software libraries	Similarity metrics, collaborative filtering

In addition to the app stores, the research field of classification and recognition of similar software projects has also increased in general. A recently conducted and published systematic literature review [3] shows the variety of approaches and motivations behind the work. In comparison, the categorization of software libraries is rarely found in published research projects. The review paper only points out the work of Escobar-Avila [5], who published an approach to automatically categorize software libraries. He manually collected the bytecode (.jar files), profiles and categories of 158 software libraries. These examined libraries are written in Java, which are published and maintained by the Apache Software Foundation. The bytecode documents, including most of the textual information in the source code, were transformed and clustered. Instead of a multi-class classification, five categories are suggested by this approach. The author points out that this goal leads to a good accuracy on recommending a list of categories, but precision and recall were rather low and in around 40% of cases, the first recommended category was the correct one. Since we are aiming for a multi-class classification with higher precision and recall, we decided on using other sources, like meta-data of the libraries.

A recent study [6] of ours has shown that a tag-based approach can provide good results for automated multi-class classification of software libraries. However, this approach is limited to tags and only applicable to a part of the software libraries. We used the same data as in this study to achieve the same goal, which is why we use this approach as a baseline.

Another, recently published study [7] shows a hybrid approach through the automated evaluation of tags and word vectors from binary files. However, the aim of the work is a multi-label classification to recommend tags for previously untagged software libraries. Therefore, even though their study is in the same subject area, it cannot be directly compared to ours. It pursues a different goal based on different data and methods. The same applies to all other studies in the systematic literature review [3]. We can imagine that a classification based on similar extracted data to that from the studies in the review could be promising when applied to software libraries.

For a better overview, the related research that followed a similar goal is summarized in the Table 1.

Another challenge is caused by applying and evaluating imbalanced data, which was also identified in an earlier study [7] on tags of MvnRepository.com. Since we were confronted with a similar problem in our experiment, we described in our previous study [6] as well as in the evaluation Section 5 all the steps taken to obtain an evaluable result.

Software Library Classification Approach

Machine learning is well suited to train models for classification tasks because they are able to identify patterns and relationships in data that are not easily identified by humans or rule-based software approaches. This capability empowers machine learning models to learn patterns to classify vast datasets, such as a wide range of software libraries. Furthermore, once these models are trained, they can adapt to changes in the data environment, which is essential considering the constant growth of software libraries. Therefore, in addition to the prediction performance of each machine learning model trained on labeled data, the speed and scalability of these models will also be examined in the later discussion.

In addition to the approaches described in related work for the classification of software libraries, we present our new machine learning approaches based on the always available software library ids (in the following referenced as id-based) as well as a hybrid solution (in the following referenced as hybrid-based) that considers ids and tags at the same time. In doing so, we evaluate and discuss the approaches by comparing them to the tag-based approach as baseline.

Tag-Based Classification (baseline)

A tag-based approach for classification of software libraries is probably obvious if tags are available for these libraries in a repository. A detailed description of this approach, the quality of the data, the preprocessing and the study results are described in a previously published paper [6]. Both the quality of the tags and the quantity of tagged software libraries play an important role when training a ML model for classification. This aspect was also described as one of the limitations of this approach. The goal would be a classification with good results based on attributes that all software libraries of the included repositories share. Tags, on the other hand, are not always available as they are often manually assigned. The two new approaches to overcome this problem are therefore described below. While the new Id-based approach is independent, the hybrid approach is based on both approaches to benefit from them.

Id-Based Classification

In this study, we refer to the names of the libraries as the unique identifier, which in the JVM and Apache Maven context consists of group-id, artifact-id, and version [8]. We have excluded the project version of the software library in advance, since these do not provide any relevance as a feature for the targeted classification. This approach focuses on classification by the group-id and the artifact-id. Before describing the approach of processing the ids, we want to summarize what constitutes these two identifiers and why they may be suitable for classification.

The structure of these two identifiers is not clearly described and strict naming conventions do not exist either. The Apache Maven project describes the group-id as a unique identifier for the project behind the software. It should be based on the Java package name conventions and begin with a reversed domain name. This can be followed by further sub-items such as division, department, project, machine, or login names [9]. The items are usually separated by dots. The artifact-id, on the other hand, is the name of the packed build without the version number [8]. This structure is illustrated by the following example:

```
group-id:artifact-id
org.mariadb.jdbc:mariadb-java-client
```

These rules are not enforced by Maven, and consequently libraries may not follow the convention. A simple, systematic filtering out of certain parts of classification-relevant words should therefore not be possible in general. Our approach still relies on an automated classification using these group-ids and artifact-ids. As part of our approach, a process is developed to extract the relevant tokens from the id. This preprocessing is described in the following Section 4. Word embeddings of the extracted token sequences are to be generated. These word embeddings can then be used to train an artificial neural network (ANN). We considered different ANN architectures and finally decided to use a recurrent neural network (RNN), which can learn to map input sentences of variable length into a fixed-dimensional vector representation [10]. Besides an RNN, we have also considered other ML algorithms and a feedforward neural network [11] (FNN). However, since the tokens from the ids are processed similarly to the tags, we excluded other ML algorithms, such as the most recently evaluated from [6]. We experimented with an FNN in addition but could not achieve a better performance. Since the structure of the FNN, which is also described in [6], expects a fixed length of the input vector, we decided to use the RNN with a variable input.

Different RNN architectures, such as Long Short-Term Memory (LSTM) [12] or Gated Recurrent Unit (GRU) [13], can be considered. In this study, we experimented with these RNN architectures, including unidirectional and bidirectional evaluation of token sequences. While

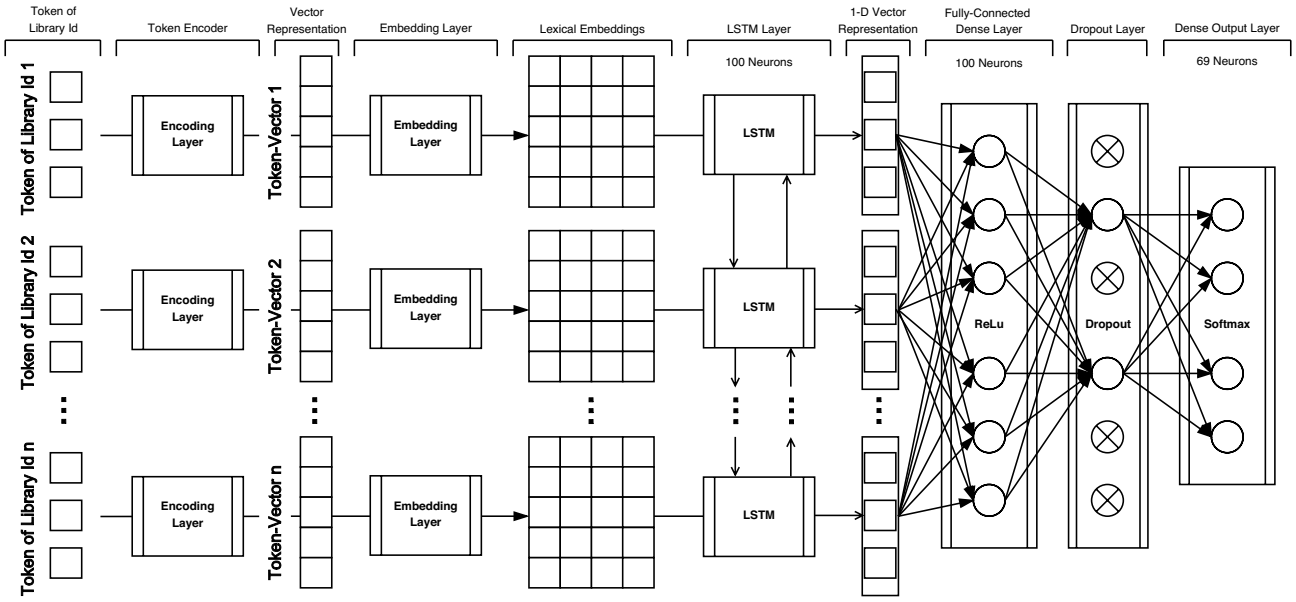

Fig. 2 Architecture of the applied RNN on the tokens of group-id and artifact-id

the differences in training results were relatively small, we opted for a bidirectional LSTM. In addition to choosing a suitable architecture, we also performed hyperparameter tuning. The resulting RNN is presented in Fig. 2.

The figure illustrates the layers placed sequentially from left to right and from top to bottom the tokenised ids of the 1 to n software libraries. To process the ids, the tokens are first loaded into an encoding layer. The resulting vectors, which represent the encoded tokens, are then converted into word embedding vectors and fed into a bidirectional LSTM layer with 100 neurons. The LSTM is followed by a fully connected dense layer with 100 neurons and a ReLu activation function. In the Figure 2, the 100 neurons of the dense layer are represented by 6 circles for better visualization. To counteract overfitting, a dropout layer follows, which randomly discards input values. For visualization purposes, we have marked neuron circles in the layer with a cross as dropouts. For those dropouts we have chosen a frequency of 20%. Finally, an output layer with 69 neurons is included, matching the 69 possible classes of the crawled dataset. Further details on the dataset and the classes are described in the following Section 4. The combination of layers, the neuron numbers, and the frequency of dropouts are the result of the hyperparameter tuning.

Hybrid Classification

The two approaches presented so far use the group-id and artifact-id of the Java software libraries and tags from the indexing platform MvnRepository.com. Complementing the two approaches, a new hybrid approach aims to combine the advantages of these two and optimizes a trained model for a higher accuracy. The idea is therefore to use the new RNN from Fig. 2, training a classification model based on the extracted tokens from the ids combined with the tag lists of the respective software libraries. The addition of ids is intended to compensate for poor tag quality or the absence of tags. However, the evaluation of the tag-based approach in the previous study [6] showed that tags, when present, produce good classification results and should always be considered.

Dataset

Software libraries for the Java Virtual Machine were crawled between May to July 2020 to create a dataset for an automated classification. For this purpose, the libraries were crawled by their id, consisting of group-id and artifact-id from the largest public repositories, listed in Section 4.1. In addition, a class mapping and improved balancing of the individual classes was carried out.

The entire process with all the efforts made to reduce the imbalance in the data is described in the previous study [6]. The mapping of the crawled 162 categories to coarser-grained classes as well as the removal of a class that was too

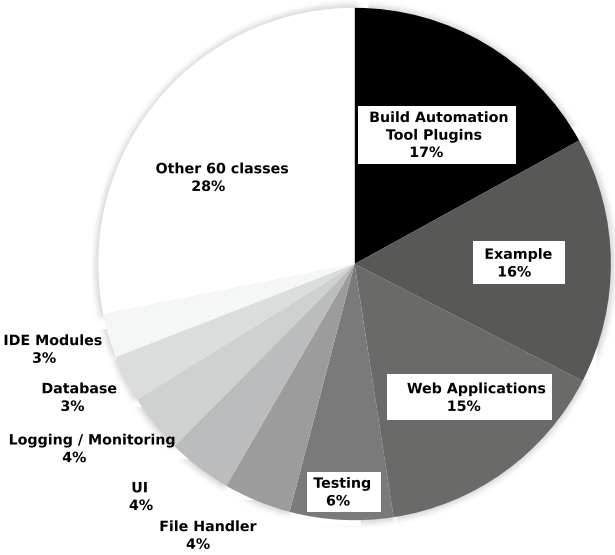

Fig. 3 Imbalanced distribution of libraries after the applied balancing steps from [6]

Description of crawled dataset

The by far largest repository was found to be Maven Central,[5] which contained slightly over 300,000 libraries. The other five selected repositories for crawling were Sonatype,[6] Spring IO,[7] Atlassian,[8] Hortonworks[9] and Wso2.[10] After merging and removing duplicates, the dataset has a size of 325,000 unique software libraries. For these crawled libraries, we accessed metadata from the online indexing service MvnRepository.com for each library. This service provides categories and tags for some libraries. While categories bundle similar libraries of a domain, tags show more coarse-grained and sometimes unique properties of a library [7]. This process of crawling is shown in Fig. 4.

For the crawled dataset, we found that not all software libraries are tagged. Furthermore, we have found that only a fraction of the libraries that have tags also have categories. Velazquez-Rodriguez and De Roover [7] point out, that tags are often missing or only a single tag is available. This observation was made based on a crawled dataset with about 3,000 tagged libraries. We were also able to identify about 75% of the crawled libraries are tagged, which corresponds to about 250,000 libraries. About 11% of all libraries are categorized. Libraries that are labeled and tagged comprise about 9% of the dataset. This is an absolute amount of around 28,600. This dataset, which we already used and published in the last study [6], is also applied in this study to ensure comparability. As described in Fig. 4, preprocessing is also carried out on the ids and a split is already made into

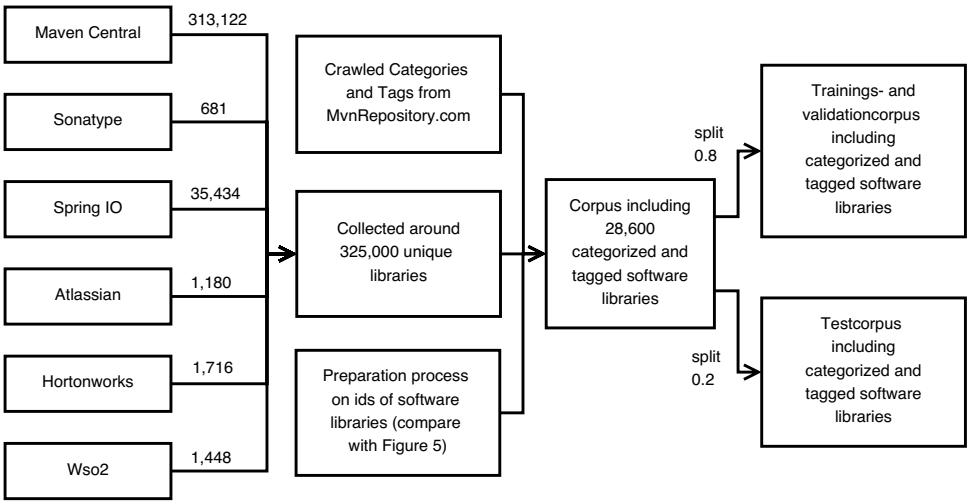

Fig. 4 Crawling Apache Maven repositories and MvnRepository.com to create software library corpora

coarse-grained is also described in detail in the last study. The 69 classes resulting from the mapping are also used as the classification target in this study. However, the imbalance could only be reduced but not eliminated. An evaluation of the distribution of classes after the applied balancing steps is shown in Fig. 3. The classes "Build Automation Tool Plugins", "Example", "Web Applications" and "Testing" still contain slightly over 50% of the labeled data.

However, in this study we did not choose an attempt to further balance the data by methods such as oversampling or undersampling, as these could bring new drawbacks, as described for example by Velez et al. [14]. Instead, by applying macro evaluation metrics and a balanced accuracy, we show below that the trained models can also deal with imbalanced data. We describe the evaluation process in section 5.

[5] https://repo1.maven.org/maven2/.
[6] https://oss.sonatype.org/content/repositories/.
[7] https://repo.spring.io/plugins-release/.
[8] https://maven.atlassian.com/content/repositories/atlassian-public/.
[9] https://repo.hortonworks.com/content/repositories/releases/.
[10] https://maven.wso2.org/nexus/content/repositories/releases/.

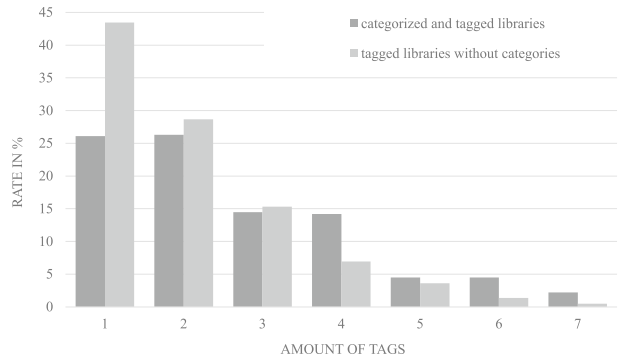

Fig. 5 Distribution of tags compared by categorized and uncategorized software libraries

a training/validation dataset and a test dataset. Details on the split process are described in Section 5. We have published the datasets applied in our evaluation on GitHub. [11]

Tags

Available tags from the indexing platform MvnRepository.com were added to the crawled software library corpus. In the process, 437 unique tags were extracted, and each software library holds usually up to 7 tags. A few outliers even have up to 13 tags assigned to them. A distribution of the tags on the categorized training and evaluation data as well as the uncategorised data is shown in Figure 5. It is noticeable that a higher percentage of libraries which are not categorized have a single tag and few libraries have more than three tags.

Additionally, we analyzed whether the uncategorised data contain different tags than the already categorized libraries. This is not the case, so we consider tags as a good basis for applying a machine learning model for automated classification.

A first analysis of the crawled dataset has shown that not all tags have the same relevance. Some tags were found to be irrelevant when viewing the crawled dataset and could be excluded for training. These are tags that occur across classes and do not contribute to the description of the functionality of a class. The excluded tags are "github", "codehaus", "apache", "experimental", "starter", "runner", "api" and "bom". We have identified these tags in an analysis of the dataset across classes and as interfering. The first three indicate where the project is hosted. The remaining tags to be excluded indicate an irrelevant status, function, or structure. By excluding these tags, we assume that an improved training result can be achieved. Even though a neural network may automatically rate the irrelevant tags as such, manual pre-filtering can exclude libraries that only have irrelevant tags from the classification. By adding further features, as described below, these insufficiently tagged software libraries can possibly still be reliably classified.

Table 2 Example of library ids from the category 'database'

group-id	artifact-id
com.h2database	h2
com.oracle.database.jdbc	ojdbc10
org.mariadb.jdbc	mariadb-java-client
org.mongodb	mongodb-driver
software.amazon.awssdk	dynamodb

Group-id and Artifact-id

The classification of software libraries by their group-id and artifact-id is novel. In contrast to the tag-based approach, the id-based approach requires additional preprocessing. The spelling of ids usually follows certain conventions, but no strict guidelines. In the following, we therefore present five software libraries of the Maven Central Repository in Table 2. These serve as examples in the following.

To achieve the best possible classification result and to obtain a model that is as robust as possible, we have carried out a automated data cleaning and described it in the form of a preprocessing process in Fig. 6. The results described later in the paper were optimized by this preprocessing. Training and validation data were repeatedly tested against the new id-based approach to achieve improvements. For comparison, the difference in results was noted in Table 3.

For example, the five software libraries presented in Table 2 were broken down into unique tokens and classified according to the procedure. Extracted tokens such as "database", "db" or "jdbc" must be identified automatically, since these are used in all the stated libraries and should be relevant for classification in the domain database.

Before we split the group-id and artifact-id into individual tokens, we start in a first step of the process with the general replacement of special characters. In addition, we removed duplicated spaces and all numbers from the text. Removing those numbers is intended to exclude resulting purely numeric tokens that represent a version number. As an example, the artifact-ids "scala-2.10-provider-plugin" and "scala-2.11-provider-plugin" with the group-id "com.atlassian.scala.plugins" can be mentioned. The version numbers define which Scala runtime is provided. However, this

[11] The datasets are available on the CCWI-repository on GitHub at https://github.com/CCWI/corpus-libsim and on https://github.com/CCWI/corpus-libsim-extended.

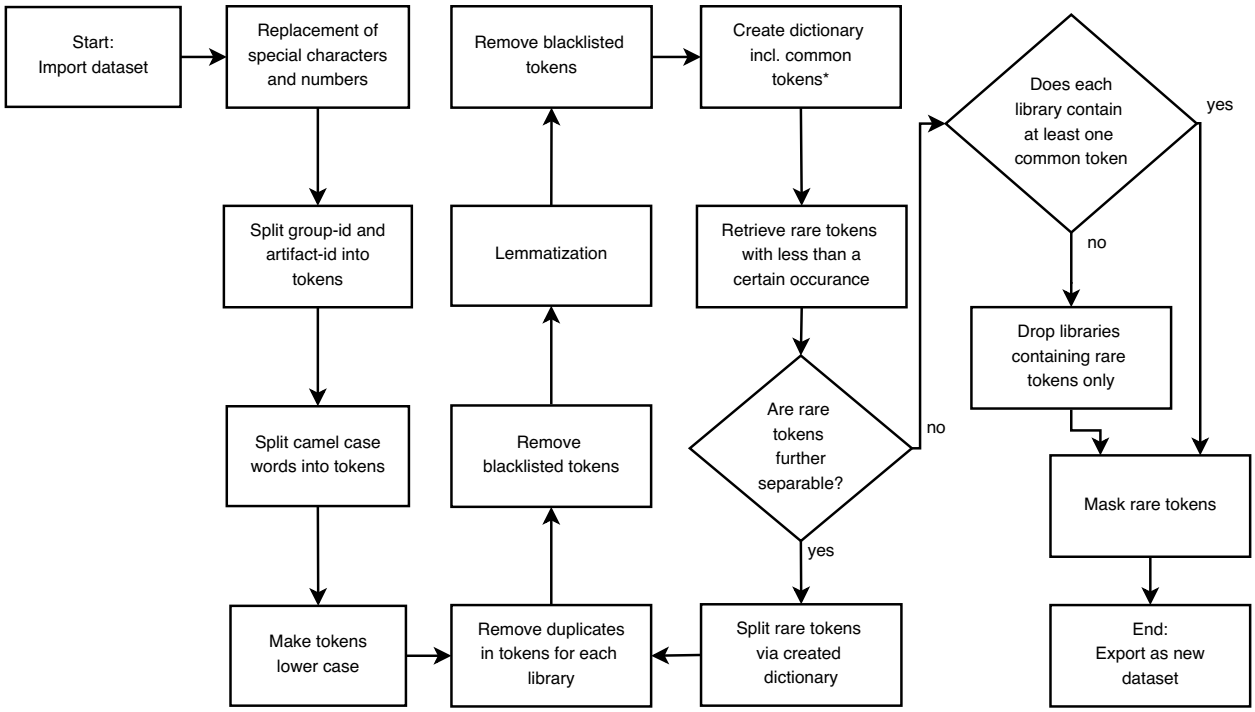

Fig. 6 Preparation process on library ids

version number is irrelevant for a classification into the targeted classes.

In the Java community, the camel case notation is commonly used and is supported by the conventions for writing ids. For text classification, this is problematic because each compound word would have its own id representation. Due to the variety of words in camel case, this would lead to a considerable amount of unique tokens and information for classification would be lost. To avoid this problem, we split the camel case words into individual tokens. After this step, all tokens are converted to lowercase letters and duplicate ones in a library id are removed. The next step can be viewed as stop word removal for software libraries. We defined a bag of blacklisted words, which do not include necessary information for the classification but do frequently occur.

First, typical entities such as the names of hosting platforms ("googlecode", "codehaus", "github'), software foundations ("apache", "eclipse") and versioning tools ("git", "svn") were removed. Company names contained in the ids, such as Oracle, Amazon, Facebook, Google, etc., were not removed, as they can provide important information for the classification. As an example from Table 2, the tokens "oracle" and "amazon" can be useful for this. Both companies are behind a respective database solution and the company names as tokens can support a classification in combination with other tokens. Furthermore, there are software libraries with unconventional names, such as "oracle:oracle", which have to be classified exclusively by the name of a company.

Another source of potentially interfering tokens in the ids could be common top-level domains such as com, edu, gov, mil, net, org or any two-letter codes identifying countries as specified in ISO standard 3166. Starting the group-id with such an abbreviation is a convention of Oracle, which is described in the JVM specification [9]. This is therefore common practice, but also not a fixed rule, as the above example ids demonstrate. However, a general exclusion of these codes is not possible, as important information could be lost in certain cases. For example, the tokens "ai" and "ml" represent country codes according to the ISO standard as well as common abbreviations for "artificial intelligence" and "machine learning" that are used in library ids. We therefore recommend a manually created dictionary that contains domain abbreviations to be removed, which are used in all libraries exclusively at the beginning of the group-id.

Next, we use the NLP library spaCy [15] for a lemmatization on each token. This allows token to be set to their basic form to facilitate classification. The token "embed" can be used as an example. Further extracted tokens like "embedding" and "embedded" are to be put into the basic form "embed" by the process. After the lemmatisation step,

it is advisable to remove any blacklisted tokens that may have emerged.

To achieve the identification of important tokens such as "database", "db" or "jdbc" in the ids, further processing steps can be applied. Frequently, important terms for the classification are concatenated within the ids, even without camel case notation. In the example from Table 2, these tokens could be extracted from the concatenated tokens "h2database", "mariadb", "mongodb" and "dynamodb". However, proper names such as "mariadb", "mongodb" and "dynamodb" may themselves already refer to a technology and can therefore be suitable for classification if these tokens occur more frequently in the dataset and especially in a certain class. Therefore, in the preparation process in Fig. 6, we propose an automated detection of common and rare tokens in the dataset and a dictionary-based split of the rare tokens if a split is possible. While common tokens are present more frequently than a certain number in the dataset, rare tokens are below a certain number. For the comparison of the different approaches, we did not optimize the dataset in this study and did not perform any splitting or filtering to ensure that the entire dataset is used. Additionally, a comparison of the id-based approach applied on different definitions of common and rare tokens in the dataset is conducted in this paper. For the comparison, in addition to training on data without split and filtering, we set tokens to be treated as common from 3 occurrences, from 5 occurrences and in another test from 10 occurrences. The applied dictionary for the split is composed of a predefined dictionary of 125,000 words from Wikipedia and a defined dictionary from the domain. The specifically defined dictionary was built up by all the found common tokens. It was ensured that the words were first split according to the domain language of the common tokens and then according to the Wikipedia dataset to avoid possible incorrect splits. Details and evaluation results are described in Section 6. The applied process in Fig. 6 also takes into account that if new tokens are found by the split process, previous steps are repeated. If no more tokens are found, all remaining rare tokens are masked. In addition, libraries without at least one common token can be removed, as these cannot be classified without further features or actions by the presented id-based approach.

Based on the preprocessing, we performed an analysis regarding the number of common tokens in the ids of software libraries. We compared classified and previously unclassified libraries to determine whether the unclassified ones contain the same common tokens. A frequency in percent of the number of common tokens without splitting and filtering rare tokens in the two datasets is shown in Fig. 7.

First, it is noticeable that many of the uncategorised libraries have at least one common token. Some libraries hold up to 8 common tokens in both datasets, a few outliers even up to 11 in the ids. It is also noticeable that about

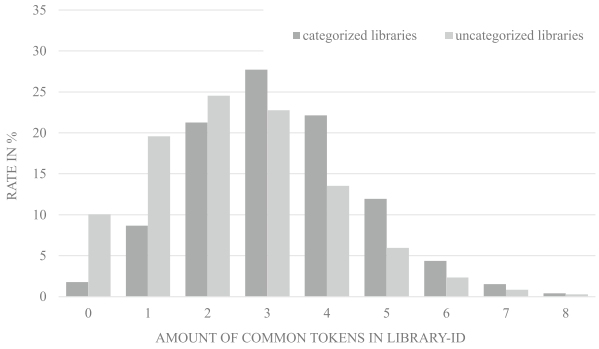

Fig. 7 Distribution of common tokens in a preprocessed dataset without splitting and filtering rare tokens compared by categorized and uncategorized software libraries

10% of the uncategorised libraries have none of the common tokens in their ids. For this 10% of the libraries, it remains to be tested as future work whether a reliable classification is possible. It may be necessary to check the tokens of the library ids separately again and to create an extended training set by manually annotating the libraries to make the trained model even more robust.

Evaluation

In our last study for the tag-based approach [6] we chose a 5-fold nested cross validation [16] for an optimized result in accuracy. At the same time, however, we were able to determine that the accuracy result does not change significantly by applying a classical training, validation and test split instead. This simplifies an accurate comparison of the approaches after all models have been trained and optimized on the same training and validation data. Furthermore, all different models were evaluated against the same test dataset, which allows a direct comparison. Therefore, we use the classical approach and split the data randomly. Initially, we reserved 20% of the data for the final evaluation. We then randomly divided the data intended for training into training data and validation data by 20%. The validation data was used to develop and optimize the resulting model. Finally, we trained the optimized neural network 5 times. For this, the random split in training and validation data was repeated.

A similar approach was also used for the second evaluation. Here, however, the same id-based approach is always applied to different datasets. The datasets differ in a varying number of tokens to categorize them as common or rare and to perform token splits and filtering of libraries. This procedure corresponds to the preprocessing presented in Fig. 6. The evaluation is intended to show one possible option for optimizing the classification results and is also discussed below.

For the evaluation in this study, we apply the metrics Precision, Recall and F1-Score, commonly used for classification problems. There are several calculation options that need to be considered. While the basic calculation is a binary variant to evaluate 2 classes, a multi-class classification involves a global calculation over the classes of true positives (TP), false negatives (FN), true negatives (TN) and false positives (FP). Without further adjustments to the calculation, this procedure is called micro-averaging [17]. However, in imbalanced datasets, the overrepresented classes are favored in the respective metrics. This is different in a calculation by macro-averaging [17]. With macro-averaging, the metrics are calculated for each class and equally merged into one overall metric. Since we have imbalanced data and perform a multi-class classification task, we have chosen the variant of macro-averaging for the respective metrics to evaluate the results. In addition to the accuracy, the balanced accuracy was calculated. The balanced accuracy considers imbalanced data to the effect that overrepresented classes are not favored over smaller classes [18].

The balanced accuracy is the average of sensitivity and specificity and can be calculated as stated in Eq. 1 [14]. Balanced accuracy is the arithmetic mean of recall for each class and is the same as the macro-averaged recall. We have calculated both metrics and our test results presented in this study show that in each test run the two metrics lead to the same results.

$$Acc_{bal} = \frac{(sensitivity+specificity)}{2} = \frac{(TP/(TP+FN)+TN/(TN+TP))}{2}. \quad (1)$$

While micro-averaging results for imbalanced data can lead to optimistic results, in our case macro-averaging results are more informative about the generalisability of the trained model. For example, looking only at micro-averaging metrics may miss the fact that the classifier assigns libraries during test from small classes to large classes. In our last study [6] we excluded this possibility by considering confusion matrices. To obtain meaningful and at the same time comparable metrics for each approach tested, we chose macro-averaging results for this evaluation. Furthermore, even though we report the accuracy for each approach, for this evaluation we prefer to use balanced accuracy to interpret the performance of each approach.

In the following, the results of the respective evaluation processes are presented and discussed.

Results

Comparison of Approaches

For comparison, we have evaluated the newly described approaches for classification by an RNN using ids, tags, and a hybrid approach. At the same time, we evaluated the tag-based approach from the last study [6] using the described procedure and compared it with the other approaches as a baseline. Unlike the other approaches, the baseline approach relies on an optimized feedforward neural network [11] (FNN). For each approach, the accuracy (Acc_{μ}), balanced accuracy (Acc_{bal}), macro-averaged recall (Rec_M), macro-averaged precision ($Prec_M$), and macro-averaged f1-score ($Fscore_M$) are calculated. A standard deviation was calculated for each result over the multiple test runs and added below the respective results. The test results for each approach based on the described evaluation procedure are summarized in Table 3. The best result in each metric is highlighted in bold.

We found the baseline approach with a balanced accuracy of 0.92 performs well despite the change in evaluation. The deviation from the results in the last study [6] is negligible. The id-based approach with an optimized RNN achieves an accuracy of 0.90, but only a balanced accuracy of 0.71. A subsequently developed hybrid approach consisting of the tag-based and id-based approach was able to further optimize the tag-based approach. The balanced accuracy is 0.94 with a standard deviation of 0.01. The additional evaluation of a pure tag-based approach based on the RNN with a balanced accuracy of 0.77 shows that it was not the neural network architecture that brought the optimisation, but the hybrid application of the data. Overall, the standard deviations for the respective metrics of all approaches are marginal and are all below 0.015.

To evaluate whether the differences in performance of the compared classifiers are significant, a non-parametric Friedman test [19] and the post-hoc Nemenyi test [20] were applied. These tests were chosen to determine if the classifiers are significantly different from each other. For the Friedman test, a null hypothesis states that there are no significant differences in performance between the classifiers, while the alternative hypothesis states that at least one classifier performs significantly better than the others. A typical p-value < 0.05 was chosen as significance level [21]. The result of the Friedman test over all approaches and their measured performance from Table 3 is the following:

statistical-value: 10.8
p-value: 0.0129

Table 3 Evaluation results of the respective models for each approach over 5 runs

Approach	Model	Measures			
		Acc_{μ}	$Acc_{\mathbf{bal}}$ / $Rec_{\mathbf{M}}$	$Prec_{\mathbf{M}}$	$Fscore_{\mathbf{M}}$
Tag-based	FNN[1]	0.9748 ± 0.0010	0.9200 ± 0.0115	0.9502 ± 0.0066	0.9291 ± 0.0104
Tag-based	RNN[2]	0.8700 ± 0.0055	0.7657 ± 0.0088	0.9425 ± 0.0145	0.8314 ± 0.0103
Id-based [3]	RNN [2]	0.9041 ± 0.0010	0.7136 ± 0.0111	0.7949 ± 0.0141	0.7403 ± 0.0110
Hybrid	RNN [2]	**0.9833** ± 0.0010	**0.9412** ± 0.0121	**0.9620** ± 0.0089	**0.9487** ± 0.0106

The best result of the respective metric in each column is highlighted in bold

[1] Feedforward neural network

[2] Recurrent neural network, described in Figure 2

[3] To justify the preprocessing process, the following poorer results were obtained in an additional test without preprocessing: $\mathbf{Acc}_{\mu} = 0.8330$ / $\mathbf{Acc}_{\mathbf{bal}} = 0.6604$ / $\mathbf{Prec}_{\mathbf{M}} = 0.7782$ / $\mathbf{Fscore}_{\mathbf{M}} = 0.7049$

Table 4 Results of the applied Nemenyi test to find significant differences between the models examined

	Tag-based (FNN)	Tag-based (RNN)	Id-based (RNN)	Hybrid (RNN)
Tag-based (FNN)	1.000	0.3549	0.3549	0.6703
Tag-based (RNN)	0.3549	1.000	0.9000	**0.0314**
Id-based (RNN)	0.3549	0.9000	1.000	**0.0314**
Hybrid (RNN)	0.6703	**0.0314**	**0.0314**	1.000

Values below the significance level of 0.05 are marked in bold

Since the p-value is below the significance level and the statistical-value is higher than the critical value of 7.815,[12] the null hypothesis can be rejected and the conclusion can be drawn that there is a significant difference among the classifiers. This means that at least one classifier performs significantly better or worse than the others. However, the Friedman test does not identify which specific classifiers differ significantly from one another. This is why a Nemenyi test was applied subsequently as a post-hoc test for more insight. It performs a pairwise comparison of each classifier result, so it identifies the performance differences among them. All p-values between the respective models are summarized in Table 4. The values that are below the significance level and refer to a significant difference between the models are marked in bold.

In addition to the significances, further details on the results can be obtained from a comparison of the confusion matrices. For each approach, we have chosen a compact representation of such a matrix over the 69 classes and compared them in Fig. 8. The x-axes of the matrices show the predicted categories for all test data for the respective model. The y-axes, on the other hand, show the actual expected categories. A diagonal line without deviating markings is therefore a desired result. The applied scale was normalized across the respective predicted classes to achieve a better representation due to the imbalanced classes.

It can be observed that the tag-based model and the id-based model have predicted one class of the models more often incorrectly. In both matrices it is the class "Web Applications". This is one of the over-represented classes in the dataset, as already described in Figure 3. In the confusion matrices, the corresponding columns are highlighted with a red border.

The training time of the models was comparatively low and can therefore be neglected. However, to give an idea of how long the training of these neural networks with the given data can take on average and how the training duration of the approaches compared, the performance over all runs including extra repeats was measured as well. It was performed on a system running the Linux distribution Ubuntu 22.04.1 LTS with an NVIDIA GeForce GTX 1080, an AMD Ryzen 9 3900XT and 64 GiB of memory. The tag-based approach with an RNN already took approx. twice as long (mean/std: 32.2 s ±5.6 s) as the same approach with an FNN (mean/std: 15.4 s ±1.83 s). While the id-based approach ran faster than the tag-based RNN (mean/std: 26.8 s ±0.45 s), the hybrid approach took the longest due to the largest amount of tokens entered (mean/std: 45.9 s ±6.29 s). While these measurements can vary significantly due to the hardware and implementation used, it should be noted that all 4 classifiers can be trained in a relatively short time by a standard mid-range gaming PC. Due to these relatively short training times, instead of retraining existing models with the

[12] The critical value was obtained using a degree of freedom of k-1, when k is the amount of classifiers, on a chi-squared distribution table.

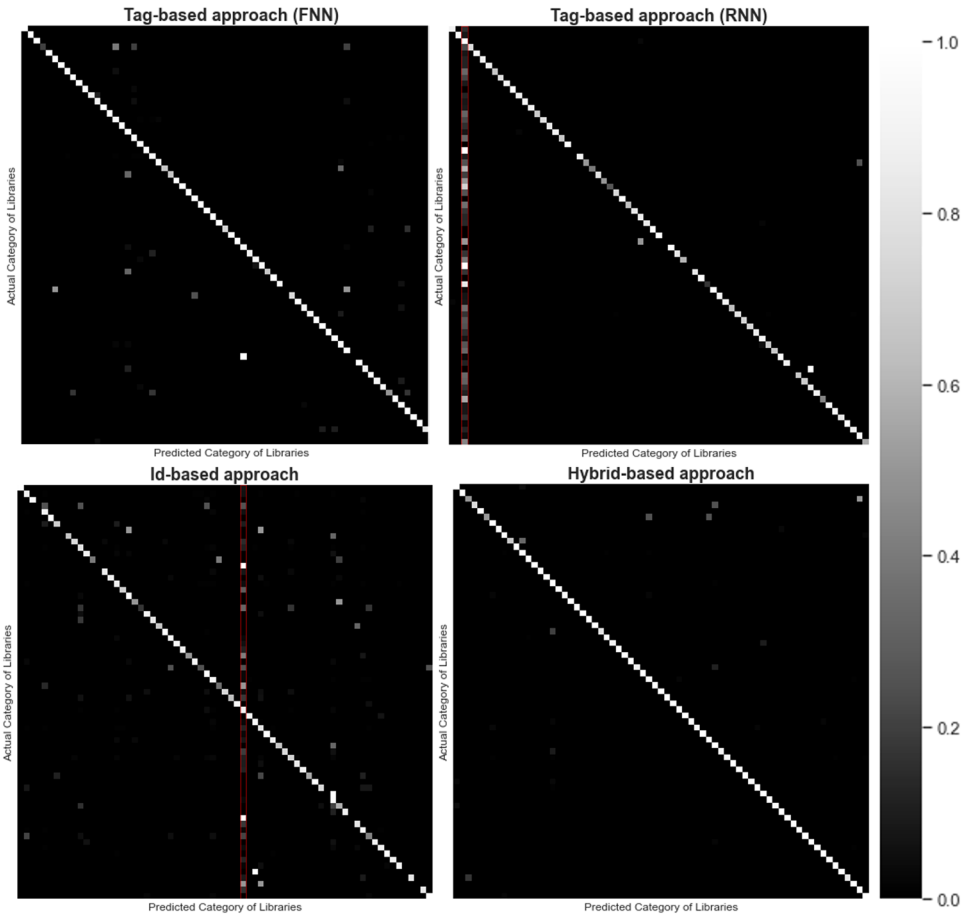

Fig. 8 Comparison of normalized confusion matrices from a trained model of each described approach

Table 5 Evaluation results of the id-based RNN approach for different dataset optimisations by rare token split and library filtering on varying $base_x$

Base	Classes	Measures			
		$\mathbf{Acc}_{\mu}$	$\mathbf{Acc}_{\mathbf{bal}}/\mathbf{Rec}_{\mathbf{M}}$	$\mathbf{Prec}_{\mathbf{M}}$	$\mathbf{Fscore}_{\mathbf{M}}$
$base_1$	69	0.9041 ± 0.0010	0.7136 ± 0.0111	0.7949 ± 0.0141	0.7403 ± 0.0110
$base_3$	69	0.9172 ±0.0021	0.7511 ± 0.0199	**0.8057** ± 0.0166	0.7703 ± 0.0182
$base_5$	69	**0.9212** ± 0.0023	**0.7683** ± 0.0136	0.7950 ± 0.0130	**0.7719** ± 0.0143
$base_{10}$	68	0.9164 ± 0.0013	0.7546 ± 0.0072	0.8046 ± 0.0116	0.7688 ± 0.0090

The best result of the respective metric in each column is highlighted in bold

risk of catastrophic forgetting, a constant retraining can also be considered as the number of data grows.

Comparison of Preprocessed Datasets

For a further evaluation, tests were carried out on the basis of optimized datasets by varying preprocessing settings. Table 5 shows the test results for the id-based RNN approach. As described before, the optimisations differ by varying definitions for common and rare tokens. The definitions for rare and common token described as base show different test results. In the following, we refer to the selected number that defines a token in the data set as common or rare as $base_x$, where x stands for the defined number of tokens. A rare token is defined as one that occurs less than x times in the data set. Tokens $\geq x$ are considered common for the dataset. For the evaluation we selected $base_1$, $base_3$, $base_5$, and $base_{10}$, which show different test outcomes when applied to the id-based RNN. The best result in each metric is highlighted in bold.

The metrics in Table 5 show that the classification results of the id-based approach can be optimized to a degree by increasing the $base_x$ for word splitting and filtering. However, the results vary. Especially the balanced accuracy could be improved significantly, while the other metrics increase only slightly. However, a general increase of the base only

brings an improvement up to a certain degree and leads to the fact that underrepresented classes can no longer be included. The evaluation of $base_{10}$ was limited to 68 classes, after one class no longer held any libraries due to the filtering. The standard deviation of the results is with less than 0.02 slightly higher than in the previous evaluation from Table 3, but still negligible. Considering the standard deviations, it can at least be concluded that the results of $base_5$ and $base_{10}$ are close to each other.

Discussion

Based on the results, it can be concluded that both an improvement of the classification results compared to the baseline was achieved and a working tag-independent approach was presented. The hybrid approach achieved better results compared to the baseline approach, based on a feedforward neural network and significantly compared to the one based on the RNN. This indicates that the quality of tags is insufficient for the classification of some libraries. Tags are insufficient if, for example, they are too generic and are used across classes. Including the tokens from the group-id and artifact-id helps the applied RNN to provide better predictions for such libraries.

The RNN trained on group-id and artifact-id achieved an accuracy of 0.90, but for balanced accuracy it achieved just about 0.71. It allows a more generic classification, independent of tags that are not always available. However, the excellent result of the tag-based approach could not be achieved. On the one hand, this could be due to the ids, which do not always contain clear or assignable tokens. On the other hand, a fuzziness in the classes or software libraries with cross-class functions could lead to the deviating predictions. Manually added tags, which are adapted to the classes, could hide this issue. An example of a software library with cross-class functions is org.apache.hbase:hbase-testing-util. This library offers utility functions for testing on the HBase database. Among others, the classes "Utilities", "Testing" and "Database" are available for the classifier. This can lead to errors in the prediction. For example, some libraries from the category "File System" were not correctly classified as "Database". The comparison of the accuracies achieved by the id-based approach and the additional consideration of the confusion matrix shows that the id-based approach works better for some classes than for others. The worst results were obtained solely for the under-represented classes. An enrichment of these small classes by, for example, selective annotation of libraries could further improve the result. We also observed that with the range of functions of some libraries and with a division of the classes as derived from MvnRepository.com, a division into classes is not always clearly possible. In addition, we found libraries on the platform that have been assigned more than one class label. An example of this is given in the following Section 8 about threads to validity. However, when the described dataset from Section 4.1 was crawled, no multi-label entries were found.

The evaluation also included the results for approaches that depend on the ids and can be optimized by more restrictive token-splitting and filtering out rare tokens in preprocessing. The improved classification results were presented in Table 5. We found for the id-based approach an optimized classification result by applying a higher $base_x$. Out of all the bases tested, the best result was achieved with $base_5$. The larger $base_{10}$ did not lead to any performance optimisation. Furthermore, an increase in the filtering of libraries by $base_{10}$ already leads to a dropout of classes that no longer contain libraries. A not further improvement of the classification results has a similar background. Already previously underrepresented classes contain even less training data after filtering. We conclude from this that optimal filtering must be determined according to needs. For the comparison of the different approaches, optimisation by filtering the software libraries was therefore not performed and $base_1$ was applied. The aim was to train models that are as robust as possible and can potentially be applied to all software libraries, and comparability with the tag-based approach is ensured by utilizing the same dataset.

Threads to Validity

The validity of the evaluation results is limited to the mapped categories, which were presented in [6] by mapping. The application of finer or more coarse-grained classes can lead to a different performance of the approaches for automated classification. The same applies to the classification into other class structures. This study refers to the class structure adapted from the classes of MvnRepository.com.

Another limitation is that the approaches are initially only applicable to software libraries in the JVM Maven context. Away from the applied dataset, the approaches need to be further evaluated. For example, the ecosystems of CRAN, PyPI or npm, which were mentioned in the introduction, need to be analyzed to conduct a cross-programming-language study on the classification of software libraries. In addition, the result may also vary when using software library ids from other repositories in the JVM context. As described, there is no binding rule regarding id names.

The data was collected between May and July 2020 and is limited to one label per software library. However, the distinction may not be definite for every software library, or the classes may be chosen in such a way that intersections arise. As an example, the library javax.inject:javax.inject [22] can be mentioned. This library was assigned the classes "Dependency Injection" and "Java Specifications". The choice of the classes in this example is because

"Dependency Injection" is function-oriented, and "Java Specifications" describes the specification of a technical standard.

Conclusion

In this study, our goal is to develop classifiers for all crawled software libraries from the JVM repositories. The study is based on around 28.600 labeled software libraries. The approach using their tokenised ids is generic and we were able to obtain a balanced accuracy of 71.36% and 76.83% by more filtering of tokens. Furthermore, we were able to optimize the classification results of the tag-based models using a hybrid approach.

This is possible by adding the tokenized ids as additional features. Training an RNN with tags and ids leads to an improvement in the results. By applying this hybrid approach, we were able to improve the balanced accuracy from 92 to 94.12%. Probably poorly tagged libraries as well as software libraries that do not hold any tags will be supplemented by the tokens from the ids, improving the classification result.

Third-party libraries from other programming languages may contain libraries with different name or id structures. Further studies and possibly further approaches are therefore necessary. At the same time, we found that other indexing platforms for software libraries in other languages tag their listed entries partly as well. It would be future work to analyze the quality and coverage of the tags on other platforms and to examine the applicability for a classification.

Another future work could be the analysis of id-tokens already described in section 4.3. An additional annotation of libraries with certain tokens could solve several problems at once. Uncategorized libraries without previous common tokens can possibly be classified more reliably. In addition, the imbalance can be reduced by enriching strongly underrepresented classes and thus the performance results of the trained models based on ids as well as the hybrid approach can be optimized.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data availability The data that support the findings of this study are openly available at (https://github.com/CCWI/corpus-libsim-extended).

Declarations

Conflict of interest During his participation in this research project, Maximilian Auch had a part-time employment relationship with Ausy Technologies Germany. Maximilian Balluff also had a part-time employment relationship with IT4IPM - IT for Intellectual Property Management GmbH. In addition to these employments, which had no direct influence on the study, the authors declare that they have no conflict of interest.

References

1. Salza P, Palomba F, Di Nucci D, de Lucia A, Ferrucci F. Third-party libraries in mobile apps. Empir Softw Eng. 2020;25(3):2341–77. https://doi.org/10.1007/s10664-019-09754-1.
2. Thung F, Lo D, Lawall J. Automated library recommendation. In: 2013 20th Working Conference on reverse engineering (WCRE), 2013; pp. 182–191. https://doi.org/10.1109/WCRE.2013.6671293.
3. Auch M, Weber M, Mandl P, Wolff C. Similarity-based analyses on software applications: a systematic literature review. J Syst Soft. 2020;168:110669
4. Yu H, Xia X, Zhao X, Qiu W. Combining collaborative filtering and topic modeling for more accurate android mobile app library recommendation. In: Mei H, editor. Proceedings of the 9th Asia-Pacific Symposium on Internetware. New York, NY: ACM Digital Library, ACM; 2017. p. 1–6. https://doi.org/10.1145/3131704.3131721.
5. Escobar-Avila J. Automatic categorization of software libraries using bytecode. In: 2015 IEEE/ACM 37th IEEE International Conference on software engineering, 2015;2:784–6.https://doi.org/10.1109/ICSE.2015.249.
6. Auch M, Balluff M, Mandl P, Wolff C. Similarity of software libraries: a tag-based classification approach. In: Quix C, editor. DATA 2021; 17–28. SCITEPRESS-Science and Technology Publications Lda, Setúbal, Portugal, 2021. https://doi.org/10.5220/0010521600170028.
7. Velázquez-Rodríguez C, De Roover C. Mutama: an automated multi-label tagging approach for software libraries on maven. In: 2020 IEEE 20th International Working Conference on source code analysis and manipulation (SCAM), 2020; 254– 258. https://doi.org/10.1109/SCAM51674.2020.00034.
8. Sanchez C. Maven—guide to naming conventions 2005. https://maven.apache.org/guides/mini/guide-naming-conventions.html#guide-to-naming-conventions-on-groupid-artifactid-and-version. Accessed 9 Nov 2021.
9. Gosling J, Joy B, Steele G, Bracha G, Buckley A, Smith D, Bierman G. The Java® Language Specification 2021. https://docs.oracle.com/javase/specs/jls/se17/html/index.html. Accessed 9 Nov 2021.
10. Sutskever I, Vinyals O, Le VQ. Sequence to sequence learning with neural networks. arXiv: org/pdf/1409.3215v3.
11. Goodfellow I, Bengio Y, Courville A. Deep learning. Adaptive computation and machine learning. Cambridge: The MIT Press; 2016.

12. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735–80. https://doi.org/10.1162/neco.1997.9.8.1735.
13. Cho K, van Merrienboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv:org/pdf/1406.1078.
14. Velez DR, White BC, Motsinger AA, Bush WS, Ritchie MD, Williams SM, Moore JH. A balanced accuracy function for epistasis modeling in imbalanced datasets using multifactor dimensionality reduction. Genet Epidemiol. 2007;31(4):306–15. https://doi.org/10.1002/gepi.20211.
15. spaCy . Industrial-strength Natural Language Processing in Python (22.11.2021). https://spacy.io/. Accessed 22 Nov 2021.
16. Varma S, Simon R. Bias in error estimation when using cross-validation for model selection. BMC Bioinform. 2006;7(1):91. https://doi.org/10.1186/1471-2105-7-91.
17. Sokolova M, Lapalme G. A systematic analysis of performance measures for classification tasks. Inform Process Manag. 2009;45(4):427–37. https://doi.org/10.1016/j.ipm.2009.03.002.
18. Brodersen KH, Ong CS, Stephan KE, Buhmann JM. The balanced accuracy and its posterior distribution. In: 2010 20th International Conference on pattern recognition (ICPR 2010), IEEE, Piscataway, NJ, 2010; 3121– 3124. https://doi.org/10.1109/ICPR.2010.764.
19. Friedman M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J Am Stat Assoc. 1937;32(200):675–701.
20. Nemenyi PB. Distribution-free multiple comparisons. Princeton University; 1963.
21. Demar J. Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res. 2006;7:1–30.
22. Maven Repository: javax.inject:javax.inject (08.12.2021). https://mvnrepository.com/artifact/javax.inject/javax.inject Accessed 08.12.2021