Using collaborative filtering to recommend Github topics

ABSTRACT

Collaborative filtering is a well-founded technique spreadly used in the recommendation system domain. During recent years, a plethora of approaches has been developed to provide the users with relevant items. Considering the open-source software (OSS) domain, GitHub has become a precious service for storing and managing software source code. To represent the stored projects in an effective manner, in 2017 GitHub introduced the possibility to classify them employing topics. However, assigning wrong topics to a given repository can compromise the possibility of helping other developers reach it and eventually contribute to its development. In this paper, we present <RECOMMENDER ACRONYM>, a recommender system to assist open source software developers in selecting suitable topics to the repositories. <RECOMMENDER ACRONYM> exploits a collaborative filtering technique to recommend libraries to developers by relying on the set of initial topics, which are currently included in the project being. To assess the quality of the approach, we exploit our previous work in this domain and validate both of them using different metrics. The results show that <RECOMMENDER ACRONYM> outperforms it in all the examined aspects.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

datasets, collaborative filtering, topic recommender

ACM Reference Format:

. 2018. Using collaborative filtering to recommend Github topics. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In recent years, the open-source software (OSS) community makes a daily usage of open source repositories to contribute their work as well as to access to projects coming from other developers. GitHub is one of the most well-known platforms that aggregate these projects and render possible the exchange of knowledge among the users. In order to aid information discovery and help developers identify projects that can be of their interest, GitHub introduced *topics*. They are words used to characterize projects, which thus can be annotated by means of lists of words that summarize projects' features. Thanks to the availability of *topics*, several applications are enabled, including the automated cataloging of GitHub repositories [**?**], further than allowing developers to explore projects by type, technology, and more.

Assigning the right topics to GitHub repositories is a crucial step that, if not properly done, can affect in a negative way their discoverability. In 2017, GitHub presented *repo-topix*, a topic suggestion tool essentially based on information retrieval techniques [**?**]. Although the mechanism works well so far and it is fully integrated with GitHub, in our opinion there is still some room for improvement, e.g., in terms of the variety of the suggested topics, novel data analysis techniques, and the investigation of new recommendation strategies.

We have already faced this problem in our previous work [**?**] by using a machine learning approach to recommend relevant topics given a README file of a repository. In this initial attempt, we are able to recommend only *featured* topics, a curated list of them provided by Github [].

In this work, we propose to extend the set of recommended items to non-featured topics by exploiting collaborative filtering, a widely spread technique in the recommendation system domain [**?**]. Given an initial set of topics coming from a GitHub project, we use repository-topic matrixes to suggest relevant topics. The work gives the following contributions:

- Considering the GitHub projects as products, we suggest relevant topics to the project given an initial list of them;
- We assess the quality of the work employing a well-defined set of metrics commonly used in the recommendation system domain i.e., sales diversity, novelty, and accuracy;
- We extend our previous work in the domain considering the entire set of topics and use it as a baseline

The rest of the work is structured as follows. Section 2 shows the issues and the potential challenges in the domain. In Section 3, we present our approach and evaluate it in Section 4. We present the results of the assessment in Section 5 and we discuss the findings. Section 6 summarizes relevant works in the field and we conclude the paper in Section 7 with possible future works.

2 BACKGROUND

Manually assigning topics can be an error-prone activity that can lead to wrongly specified tags. Over the last years, several attempts have been made to *classify* GitHub projects by automatically inferring appropriate topics. In the context of data mining, *classification* is one of the critical operations that are used to dig deep into available data for gaining knowledge and for identifying repetitive patterns [**?**].

In [**?**] the authors present an approach based on *topic modeling* techniques to create categories of GitHub projects. Manual interventions are needed to refine initial sets of categories, which are identified by an LDA-GA technique, that combines two algorithms: Latent Dirichlet Allocation (LDA) and Genetic Algorithm (GA) [**?**].

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

The approach proposed in [?] is unsupervised, meaning that the categories of the catalogue being identified are not known ex-ante.

In a GitHub blog post [?] the author presents *repo-topix*, a tool to recommend topics for GitHub repositories. Such a tool combines NLP standard techniques to find an initial set of topics, by parsing the README files and the textual content of a repository e.g., the repository's description. Then, they weight the results with the TF-IDF scheme and remove "bad" topics using a regression model. Using this refined list, repo-topix computes a custom version of Jaccard Distance to identify additional similar topics. To assess the quality of the framework, they made a rough evaluation based on ROUGE-1 metrics, an n-gram overlap metric that counts the number of overlapping units between the suggested topics and the repository description. Unfortunately, in [?] the author discusses an approximation of the repo-topix accuracy, without providing the reader with the complete dataset that was used and the source code of the developed tool.

With the aim of contributing the resolution of the problem of recommending GitHub topics, in the next section we propose to use item-based collaborative filtering to recommend relevant topics. The challenges that we had to cope with for evaluating its performance are mainly the following ones:

▹ *Dataset definition:* the creation of the datasets to be used for evaluating the approach being proposed and comparing it with some baseline is a daunting task: repositories might be moved, heavily changed or even deleted during the initial creation. Thus, the crawling activity can be negatively affected by these continuous changes and lead to lack of data, and poor topic coverage. GHTorrent[1] tries to mitigate this issue by offering daily dumps of the repositories' metadata. However, this kind of data might not be enough or even appropriate (e.g., source code is not available in GHTorrent dumps) to properly classify an entire repository. Even considering directly GitHub data can be difficult: GitHub limits the total number of requests per hour to 5,000 for authenticated users and 60 for unauthorized requests. Considering all these constraints, building a suitable dataset represents a real challenge to be managed carefully.

▹ *Topics distribution:* although tags can be assigned only by the owners of GitHub repositories, users can potentially wrongly specify topics or introduce information overload by inserting too many elements. Thus, creating a reliable ground truth to assess the classification performance of the proposed approach represents another relevant difficulty.

3 PROPOSED APPROACH

In this section we describe <RECOMMENDER ACRONYM>, a system for providing developers with GitHub topics related recommendations. More specifically, <RECOMMENDER ACRONYM> is a *recommender system* [?] that encodes the relationships among various OSS artifacts by means of a semantic graph and utilizes a collaborative filtering technique [?] to recommend GitHub topics. Such a technique has been originally developed for e-commerce systems to exploit the relationships among users and products to predict the missing ratings of prospective items [?]. The technique

[1]http://ghtorrent.org/

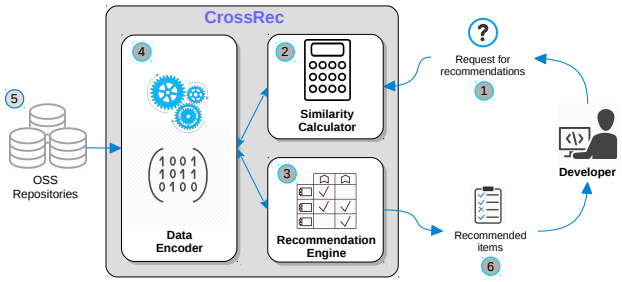

Figure 1: Overview of the <RECOMMENDER ACRONYM> Architecture.

is based on the premise that *"if users agree about the quality or relevance of some items, then they will likely agree about other items"* [?]. Our approach exploits this premise to solve the problem of library recommendation [?]. Instead of recommending goods or services to customers, we recommend third-party libraries to projects using an analogous mechanism: *"if projects share some libraries in common, then they will probably share other common libraries."*

To this end, the architecture of <RECOMMENDER ACRONYM> is shown in Fig. 1, and consists of the software components supporting the following activities:

- *Representing the relationships* among projects and topics retrieved from existing repositories;
- *Computing similarities* to find projects, which are similar to that under development; and
- *Recommending topics* to projects using a collaborative-filtering technique.

In a typical usage scenario of <RECOMMENDER ACRONYM>, we assume that a developer is creating a new system, in which she has already included some topics, or else is evolving an existing system. As shown in Fig. 1, the developer interacts with the system by sending a request for recommendations ①. Such a request contains a list of topics that are already included in the project the developer is working on. The `Data Encoder` ④ collects *background data* from OSS repositories ⑤, represents them in a graph, which is then used as a base for other components of <RECOMMENDER ACRONYM>. The `Similarity Calculator` module ② computes similarities among projects to find the most similar ones to the given project. The `Recommendation Engine` ③ implements a *collaborative-filtering* technique [?],[?], it selects top-k similar projects, and performs computation to generate a ranked list of *top-N* topics. Finally, the recommendations ⑥ are sent back to the developer.

Background data has been collected GitHub [?]. The current version of <RECOMMENDER ACRONYM> [?] supports data extraction from GitHub, even though the support for additional platforms is under development.

In the following, the components `Data Encoder`, `Similarity Calculator`, and `Recommendation Engine` are singularly described.

3.1 Data Encoder

A recommender system for online services is based on three key components, namely *users*, *items*, and *ratings* [?],[?]. A *user-item*

ratings matrix is built to represent the mutual relationships among the components. Specifically, in the matrix a user is represented by a row, an item is represented by a column and each cell in the matrix corresponds to a rating given by a user for an item [?]. For topic recommendation, instead of users and items, there are projects and topics, and a project may include various topics to better describe the project. We derive an analogous user-item ratings matrix to represent the *project-library inclusion* relationships, denoted as ∋. In this matrix, each row represents a project and each column represents a topic. A cell in the matrix is set to 1 if the topic in the column is included in the project specified by the row, it is set to 0 otherwise. For the sake of clarity and conformance, we still denote this as a user-item ratings matrix throughout this paper.

For explanatory purposes, we consider a set of four projects $P = \{p_1, p_2, p_3, p_4\}$ together with a set of topics $L = \{topic_1$=*machine-learning*; $topic_2$=*javascript*; topiclib$_3$=database; $topic_4$=*web*; $topic_5$=*algorithm*$\}$. By extracting the list of defined topics of the projects in P, we discovered the following inclusions: $p_1 \ni topic_1, topic_2$; $p_2 \ni topic_1, topic_3$; $p_3 \ni topic_1, topic_3, topic_4, topic_5$; $p_4 \ni topic_1, topic_2, topic_4, topic_5$. Accordingly, the user-item ratings matrix built to model the occurrence of the topic is depicted in Fig. 2.

	$topic_1$	$topic_2$	$topic_3$	$topic_4$	$topic_5$
p_1	1	1	0	0	0
p_2	1	0	1	0	0
p_3	1	0	1	1	1
p_4	1	1	0	1	1

Figure 2: An example of a user-item ratings matrix to model the inclusion of topics in GitHub repositories.

3.2 Similarity Calculator

The `Recommendation Engine` of <RECOMMENDER ACRONYM> works by relying on an analogous user-item ratings matrix. To provide inputs for this module, the first task of <RECOMMENDER ACRONYM> is to perform similarity computation on its input data to find the most similar projects to a given project. In this respect, the ability to compute the similarities among projects has an effect on the recommendation outcomes. Nonetheless, computing similarities among software systems is considered to be a difficult task [?]. In addition, the diversity of artifacts in OSS repositories as well as their cross relationship makes the similarity computation become even more complicated. In OSS repositories, both humans (i.e., developers and users) and non-human actors (such as repositories, and libraries) have mutual dependency and implication on the others. The interactions among these components, such as developers commit to or star repositories, or projects include libraries, create a tie between them and should be included in similarity computation.

We assume that a representation model that addresses the semantic relationships among miscellaneous factors in the OSS community is beneficial to project similarity computation. To this end, we consider the community of developers together with OSS projects, topics, and their mutual interactions as an *ecosystem*. We derive a

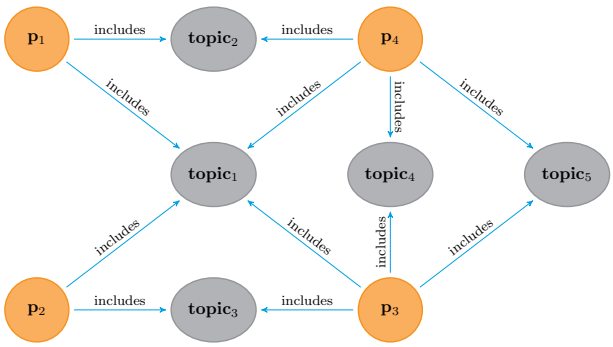

Figure 3: Graph representation for projects and libraries.

graph-based model to represent different kinds of relationships in the OSS ecosystem, and eventually to calculate similarities. In the context of mining OSS repositories, the graph model is a convenient approach since it allows for flexible data integration and numerous computation techniques. By applying this representation, we are able to transform the set of projects and topics shown in Fig. 2 into a directed graph as in Fig. 3. We adopted our proposed CrossSim approach [?],[?] to compute the similarities among OSS graph nodes. It relies on techniques successfully exploited by many studies to do the same task [?],[?]. Among other relationships, two nodes are deemed to be similar if they point to the same node with the same edge. By looking at the graph in Fig. 3, we can notice that p_3 and p_4 are highly similar since they both point to three nodes $topic_1, topic_4, topic_5$. This reflects what also suggested in a previous work by McMillan et al. [?], i.e., similar projects implement common pieces of functionality by using a shared set of libraries.

Using this metric, the similarity between two project nodes p and q in an OSS graph is computed by considering their feature sets [?]. Given that p has a set of neighbor nodes $(topic_1, topic_2, .., topic_l)$, the features of p are represented by a vector $\overrightarrow{\phi} = (\phi_1, \phi_2, .., \phi_l)$, with ϕ_i being the weight of node $topic_i$. It is computed as the *term-frequency inverse document frequency* value as follows:

$$\phi_i = f_{topic_i} \times log(\frac{|P|}{a_{topic_i}}) \tag{1}$$

where f_{topic_i} is the number of occurrence of $topic_i$ with respect to p, it can be either 0 and 1 since there is a maximum of one $topic_i$ connected to p by the edge *includes*; $|P|$ is the total number of considered projects; a_{topic_i} is the number of projects connecting to $topic_i$ via the edge *includes*. Eventually, the similarity between p and q with their corresponding feature vectors $\overrightarrow{\phi} = \{\phi_i\}_{i=1,..,l}$ and $\overrightarrow{\omega} = \{\omega_j\}_{j=1,..,m}$ is computed as given below:

$$sim(p, q) = \frac{\sum_{t=1}^{n} \phi_t \times \omega_t}{\sqrt{\sum_{t=1}^{n}(\phi_t)^2} \times \sqrt{\sum_{t=1}^{n}(\omega_t)^2}} \tag{2}$$

where n is the cardinality of the set of libraries that p and q share in common [?]. Intuitively, p and q are characterized by using vectors in an n-dimensional space, and Eq. 2 measures the cosine of the angle between the two vectors.

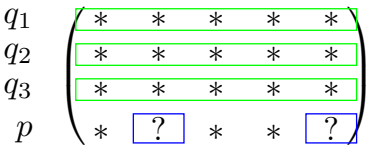

Figure 4: Computation of missing ratings using the user-based collaborative-filtering technique [?].

3.3 Recommendation Engine

Phuong ▶*Please reprhare this section*◀ The representation using a user-item ratings matrix allows for the computation of missing ratings [?],[?]. Depending on the availability of data, there are two main ways to compute the unknown ratings, namely *content-based* [?] and *collaborative-filtering* [?] recommendation techniques. The former exploits the relationships among items to predict the most similar items. The latter computes the ratings by taking into account the set of items rated by similar customers. There are two main types of collaborative-filtering recommendation: *user-based* [?] and *item-based* [?] techniques. As their names suggest, the user-based technique computes missing ratings by considering the ratings collected from similar users. Instead, the item-based technique performs the same task by using the similarities among items [?].

In the context of <RECOMMENDER ACRONYM>, the term *rating* is understood as the occurrence of a library in a project and computing missing ratings means to predict the inclusion of additional libraries. The project that needs prediction for library inclusion is called the *active project*. By the matrix in Fig. 4, p is the active project and an asterisk ($*$) represents a known rating, either 0 or 1, whereas a question mark (?) represents an unknown rating and needs to be predicted.

Given the availability of the cross-relationships as well as the possibility to compute similarities among projects using the graph representation, we exploit the user-based collaborative-filtering technique as the engine for recommendation [? ?]. Given an active project p, the inclusion of libraries in p can be deduced from projects that are similar to p. The process is summarized as follows:

- Compute the similarities between the active project and all projects in the collection;
- Select *top-k* most similar projects; and
- Predict ratings by means of those collected from the most similar projects.

The rectangles in Fig. 4 imply that the row-wise relationships between the active project p and the similar projects q_1, q_2, q_3 are exploited to compute the missing ratings for p. The following formula is used to predict if p should include l, i.e., $p \ni l$ [?]:

$$r_{p,l} = \overline{r_p} + \frac{\sum_{q \in topsim(p)} (r_{q,l} - \overline{r_q}) \cdot sim(p,q)}{\sum_{q \in topsim(p)} sim(p,q)} \quad (3)$$

where $\overline{r_p}$ and $\overline{r_q}$ are the mean of the ratings of p and q, respectively; q belongs to the set of *top-k* most similar projects to p, denoted as $topsim(p)$; $sim(p,q)$ is the similarity between the active project and a similar project q, and it is computed using Equation 2.

4 EVALUATION

This section describes the planning of our evaluation, having the *goal* of evaluating the performance of the proposed approach. In Section 4.1, we introduce the dataset exploited in our evaluation. The evaluation methodology and metrics are presented in Section 4.2. Finally, Section 4.3 describes the research questions.

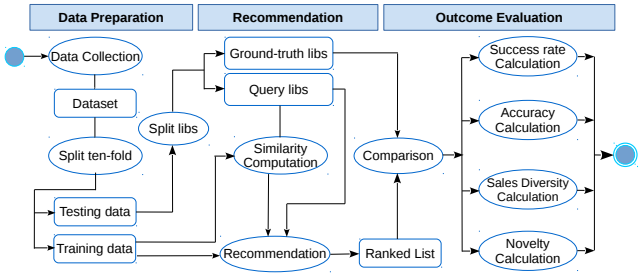

Figure 5: Evaluation Process.

The evaluation process is depicted in Fig. 5 and it consists of three consecutive phases, i.e., *Data Preparation*, *Recommendation*, and *Outcome Evaluation*. We start with the *Data Preparation* phase by creating a dataset from GitHub projects. The dataset is then split into training and testing sets. In the *Recommendation* phase, given a project in the testing set, a portion of its topics is extracted as ground-truth data, and the remaining is used as query to compute similarity and recommendations. Finally, by the *Outcome Evaluation* phase, we compare the recommendation results with those stored as ground-truth data to compute the quality metrics. All the aforementioned phases are detailed in the rest of this section.

4.1 Dataset Extraction

To evaluate the approach, we reuse the same dataset employed for our previous work [?].

By means of the GitHub API [?] we *randomly* collected a dataset consisting of 6,258 repositories that use 15757 topics. Using a project's name as a query, we can retrieve different types of information such as commits, issues, and name of the repository's owner, and the list of topics. By using the GitHub query language [?], we applied the following query filter:

$$Qf = "is:featured\ topic:t\ stars:100..80000\ topics:>=2" \quad (4)$$

to consider only GitHub repositories having a number of stars between 100 and 80,000, and tagged with at least two topics. We set such a filter after several query refinements, as the chosen featured topics have a different degree of popularity. For instance, the most starred project of the *3d* topic has around 53,000 stars; reversely, some other topics have top rank projects that do not reach 1,000 stars. Thus, we tried to select the best query filter to include a sufficient number of repository for each topic. We also impose an additional filter on the topics frequencies, to prune unpopular ones from the dataset. We made this choice as our boundaries include also non-featured topics in the comparison.

GitHub developers use stars as a voting mechanism to foster the popularity of a certain project []. Through this system, each user can support her/his favorite projects available on the platform. Forking is another index related to the quality of the project. This

feature is typically employed as a starting point for a new project [**?**]. Furthermore, there is a strong correlation between forks and stars [**?**]. In this sense, we suppose that a project with a high number of stars means that it gets attention from the OSS community and thus being suitable to identify popular repositories [**? ?**].

4.2 Evaluation Methodology and Metrics' Definition

To assess the performance of <RECOMMENDER ACRONYM> proposed approach, we applied ten-fold cross-validation, considering every time 9 folds (each one contains 625 projects) for training and the remaining one for testing. For every testing project p, a half of its topics are *randomly* taken out and saved as ground truth data, let us call them $GT(p)$, which will be used to validate the recommendation outcomes. The other half are used as testing libraries or query, which are called ***te***, and serve as input for `Similarity Computation` and `Recommendation`. The splitting mimics a real development process where a developer has already included some topics in the current project, i.e., ***te*** and waits for recommendations, that means additional topics to be incorporated. A recommender system is expected to provide her with the other half, i.e., $GT(p)$.

There are several metrics available to evaluate a ranked list of recommended items [**?**]. In the scope of this paper, *success rate*, *accuracy*, *sales diversity*, and *novelty* have been used to study the systems' performance as already proposed by Robillard et al. [**?**] and other studies [**?**],[**?**]. For a clear presentation of the metrics considered during the outcome evaluation, let us introduce the following notations:

- N is the cut-off value for the ranked list;
- k is the number of neighbor projects exploited for the recommendation process;
- For a testing project p, a half of its libraries are extracted and used as the ground-truth data named as $GT(p)$;
- $REC(p)$ is the *top-N* libraries recommended to p. It is a ranked list in descending order of real scores;
- If a recommended library $l \in REC(p)$ for a testing project p is found in the ground truth of p (i.e., $GT(p)$), hereafter we call this as a library *match* or *hit*.

If $REC_N(p)$ is the set of top-N items and $match_N(p) = GT(p) \bigcap REC_N(p)$ is the set of items in the *top-N* list that match with those in the ground-truth data, then the metrics are defined as follows.

Success rate@N. Given a set of testing projects P, this metric measures the rate at which a recommender system returns at least a topic match among *top-N* items for every project $p \in P$ [**?**]:

$$success\ rate@N = \frac{count_{p \in P}(|match_N(p)| > 0)}{|P|} \tag{5}$$

where the function *count()* counts the number of times that the boolean expression specified in its parameter is *true*.

Accuracy. Accuracy is considered as one of the most preferred *quality indicators* for Information Retrieval applications [**?**]. However, *success rate@N* does not reflect how accurate the outcome of a recommender system is. For instance, given only one testing project, there is no difference between a system that returns 1 topic match out of 5 and another system that returns all 5 topic matches, since *success rate@5* is 100% for both cases (see Eq. (5)). Thus, given a list of *top-N* libraries, *precision@N* and *recall@N* are utilized to measure the *accuracy* of the recommendation results. *precision@N* is the ratio of the *top-N* recommended topics belonging to the ground-truth dataset, whereas *recall@N* is the ratio of the ground-truth topics appearing in the N recommended items [**?**],[**?**],[**?**]:

$$precision@N = \frac{|match_N(p)|}{N} \tag{6}$$

$$recall@N = \frac{|match_N(p)|}{|GT(p)|} \tag{7}$$

Sales Diversity. This term originates from the business domain where it is important to improve the coverage as also the distribution of products across customers, thereby increasing the chance that products will get sold by being introduced [**?**]. Similarly, in the context of topic recommendation, *sales diversity* indicates the ability of the system to suggest to projects as much topics as possible, as well as to disperse the concentration among all items, instead of focusing only on a specific set of topics [**?**]. In the scope of this paper, *catalog coverage* and *entropy* are utilized to gauge *sales diversity*. Let L be the set of all topic available for recommendation, $\#rec(l)$ denote the number of projects getting library l as a recommendation, i.e., $\#rec(l) = count_{p \in P}(|REC_N(p) \ni l|)$, $l \in L$, and $total$ denote the number of recommended items across all projects.

Catalog coverage measures the percentage of topic being recommended to projects:

$$coverage@N = \frac{\left|\cup_{p \in P} REC_N(p)\right|}{|L|} \tag{8}$$

Entropy evaluates if the recommendations are concentrated on only a small set or spread across a wide range of topic:

$$entropy = -\sum_{l \in L}\left(\frac{\#rec(l)}{total}\right) ln\left(\frac{\#rec(l)}{total}\right) \tag{9}$$

Novelty. In business, the *long tail effect* is the fact that a few of the most popular products are extremely popular, while the rest, so-called the long tail, is obscure to customers [**?**]. Recommending products in the long tail is beneficial not only to customers but also to business owners [**?**]. Given that the logarithmic scale is used instead of the decimal one on the x-Axis of Fig. **??**, Fig. **??**, and Fig. **??**, the long tail effect can be spotted there as a few topic are very popular by being included in several projects, whereas most of the topics appear in fewer projects. Specifically, in `D1` just 15 topics are used by more than 200 projects, whereas 14, 876 topics are used by no more than 10 repositories. Juri ▶*CHANGE VALUES*◀ In `D2` there are 876 libraries, accounting for 63.85% of the total number, being used by no more than 10 projects. Similarly, in `D3`, 88.24% of the dependencies corresponding to 49, 799 libraries are used by no more than 10 projects. When recommending a library, *novelty* measures if a system is able to pluck libraries from the "long tail" to expose them to projects. This might help developers come across *serendipitous* libraries [**?**], e.g., those that are seen by chance but turn to be useful for their current project. To quantify *novelty*, we utilize *expected popularity complement* (EPC) which is defined in the following equation [**?**]:

$$EPC@N = \frac{\sum_{p \in P} \sum_{r=1}^{N} \frac{rel(p,r) * [1 - pop(REC_r(p))]}{log_2(r+1)}}{\sum_{p \in P} \sum_{r=1}^{N} \frac{rel(p,r)}{log_2(r+1)}} \quad (10)$$

where $rel(p, r) = |GT(p) \bigcap REC_r(p)|$ represents the relevance of the library at the r position of the *top-N* list to project p, $rel(p, r)$ can either be 0 or 1; $pop(REC_r(p))$ is the popularity of the library at the position r in the *top-N* recommended list. It is computed as the ratio between the number of projects that receive $REC_r(p)$ as a recommendation and the number of projects that receive the most ever recommended library as a recommendation. Equation 10 implies that the more unpopular libraries a system recommends, the higher the EPC value it obtains and vice versa.

We use the Wilcoxon ranked sum test (considering one data point for each fold of the cross-validation) with a significance level of $\alpha = 5\%$, and the Cliff's delta (d) effect size measure [**?**]. Due to multiple tests being performed, we adjust p-values using the Holm's correction [**?**].

4.3 Research Questions

By performing the evaluation, we aim at addressing the following research questions:

- RQ1: How does the variation of training data impact on the prediction performance? To answer this question, we varied the dimension of the dataset to include more training data. In particular, we assess the quality of three different datasets to find out the best one in terms of success rate.
- RQ2: Is the approach able to provide consistent recommendations? We compute the metrics given in Section 4.2 to measure the relevance of our suggested topics considering the distribution of the considered repositories.

We study the experimental results in the next section by referring to these research questions.

5 RESULTS

This section discusses the findings of the qualitative assessment. To address the formulated research questions, we perform three different experiments i.e., EXP1, EXP2, and EXP3. In the first one, we measure the performance of the MNB approach considering also not-featured topics (see 5.1). EXP2 considers the proposed approach in this work and the results are shown in Section 5.2. To answer ..., we combine the two approaches and evaluate them in Section 5.3.

5.1 MNB evaluation

5.2 <RECOMMENDER ACRONYM> evaluation

5.3 Chain of approaches evaluation

6 RELATED WORK

This section discusses both *(i)* approaches based on collaborative filtering techniques in recommending activity and *(ii)* works that mine GitHub projects.

6.1 Recommends item by means of collaborative filtering

Amazon [**?**] proposes an item-to-item recommendation system to suggest relevant products to the final user.

6.2 Recommending OSS using GitHub topics

7 CONCLUSIONS AND FUTURE WORK

conclusion