TopFilter: An Approach to Recommend Relevant GitHub Topics

Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong Nguyen, Riccardo Rubei
Università degli studi dell'Aquila, Via Vetoio 2, 67100 – L'Aquila, Italy
{juri.dirocco,davide.diruscio,phuong.nguyen}@univaq.it,{claudio.disipio,riccardo.rubei}@graduate.univaq.it

ABSTRACT

Background: In the context of software development, GitHub has been at the forefront of platforms to store, analyze and maintain a large number of software repositories. Topics have been introduced by GitHub as an effective method to annotate stored repositories. However, labeling GitHub repositories should be carefully conducted to avoid adverse effects on project popularity and reachability. *Aims:* We present TopFilter, a novel approach to assist open source software developers in selecting suitable topics for GitHub repositories being created. *Method:* We built a project-topic matrix and applied a syntactic-based similarity function to recommend missing topics by representing repositories and related topics in a graph. The ten-fold cross-validation methodology has been used to assess the performance of TopFilter by considering different metrics, i.e., success rate, precision, recall, and catalog coverage. *Result:* The results show that TopFilter recommends good topics depending on different factors, i.e., collaborative filtering settings, considered datasets, and pre-processing activities. Moreover, TopFilter can be combined with a state-of-the-art topic recommender system (i.e., MNB network) to improve the overall prediction performance. *Conclusion:* Our results confirm that collaborative filtering techniques can successfully be used to provide relevant topics for GitHub repositories. Moreover, TopFilter can gain a significant boost in prediction performances by employing the outcomes obtained by the MNB network as its initial set of topics.

CCS CONCEPTS

• **Software and its engineering** → *Search-based software engineering*; • **Computing methodologies** → *Cross-validation*; • **Information systems** → Retrieval effectiveness.

KEYWORDS

Recommender systems, GitHub topics recommendation, Collaborative filtering

ACM Reference Format:
Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong Nguyen, Riccardo Rubei. 2020. TopFilter: An Approach to Recommend Relevant GitHub Topics. In *ESEM '20: ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '20), October 8–9, 2020, Bari, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3382494.3410690

ESEM '20, October 8–9, 2020, Bari, Italy
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7580-1/20/10…$15.00
https://doi.org/10.1145/3382494.3410690

1 INTRODUCTION

In recent years, the software developer community has intensively exploited open source repositories during their daily activities. GitHub has become one of the most popular platforms that aggregate these projects and support collaborative development activities [?]. GitHub recently introduced the possibility to tag repositories employing *topics*[1] intending to foster the popularity and promote information discovery about available projects.[2] Topics are terms used to characterize repositories and to facilitate their discoverability from software developers who are searching for existing projects providing them with some reusable features.

Assigning repositories with wrong topics can compromise their popularity and reachability. To deal with such problems, in 2017 GitHub introduced a mechanism named Repo-Topix to suggest topics by relying on various information retrieval techniques [?]. To improve Repo-Topix and to explore additional recommendation strategies, in our recent work (named MNBN hereafter) [?], we proposed a Multinomial Naïve Bayesian network to recommend relevant topics starting from the README file(s) of the repository of interest. However, such a tool can recommend only *featured* topics, i.e., a set of topics, which are curated by GitHub.[3]

In this paper, we propose TopFilter, a recommender system that extends the recommendation capabilities of the MNBN approach to non-featured topics by exploiting a collaborative filtering technique, which is widely used in the recommender system domain [?]. Given an initial set of topics already assigned to the GitHub repository of interest, we encode it in a graph-based structure to represent the mutual relationships between repositories and topics. From this, a project-topic matrix is created by following the typical user-item structure used in existing collaborative filtering applications. Then, we compute a similarity function based on featured vectors to recommend the most similar topics.

We evaluate TopFilter's prediction performances by changing different parameters. Moreover, as a direct comparison between TopFilter and MNBN is not possible due to their internal construction, we used a well-defined set of metrics used in the literature to evaluate them by considering different datasets. More importantly, we showed that by augmenting MNBN with TopFilter, we are able to provide more relevant topics. In this sense, we substantially improve the performance of the original MNBN approach.

The contributions of this paper are as follows:

- By considering GitHub topics as a product to recommend, we improve repositories' popularity by suggesting a list of relevant topics;

[1] https://help.github.com/en/github/administering-a-repository/classifying-your-repository-with-topics
[2] Hereafter the terms GitHub *"projects"* and *"repositories"* are used interchangeably.
[3] https://github.com/topics

- We assess the quality of the work employing a well-defined set of metrics commonly used in the recommendation system domain, i.e., success rate, accuracy, and catalog coverage;
- Considering a well-founded approach, we improve it by recommending an extended set of topics.

The paper is structured into the following sections. Section 2 presents the context of this work by means of a motivating example. In Section 3, we introduce the TopFilter approach, with its evaluation being presented in Section 4. Section 5 discusses relevant findings, and the related threats to validity are discussed in Section 6. Related work is summarized in Section 7. Finally, we conclude the paper and discuss future work in Section 8.

2 MOTIVATION AND BACKGROUND

GitHub is one of the most used development services that includes version control systems (i.e., git) plus social and collaborative features (e.g., bug tracking, contribution requests, task management, and wikis). At the time of writing this paper, GitHub counts more than 40 million users and over 100 million repositories. Because of this enormous amount of data, the availability of reusable projects might be compromised if they cannot be suitably discovered. In recent years, GitHub introduced a mechanism based on topics to explore repositories. The managed GitHub repositories are being continuously monitored and assigned with topics to improve their organization. Moreover, repositories are periodically analyzed to extract the most popular and active topics (i.e., *featured topics*[4]). Thus, users can observe the community's trend by consulting such a public list. In the beginning, this activity was entirely done by humans (i.e., project contributors) who label the repository according to their knowledge, feeling and belief. In the literature, there are a plenty of approaches that mine and exploit available data to analyze repositories. Nevertheless, a few of them cope with the topic recommendation task, which can be crucial in the project's development initial phase. Figure 1 shows an explanatory repository with related topics. By this simple snapshot, a GitHub user can figure out that the *SpaceshipGenerator*[5] repository makes use of *Blender-scripts* (i.e., a *python* 3d modeling library) for *procedural-generation* of *3d* spaceships from a random seed.

As previously mentioned, the MNBN approach takes as input README file(s) of a given repository to recommend the related featured topics, which can be assigned to it. To conceive MNBN [**?**], we have adopted standard techniques employed in the ML domain, i.e., textual engineering, feature extraction, and training phase. By relying on the multinomial probability distribution, the approach can extract relevant information from README file(s) and suggest a set of topics. Table 1 shows an example of the outcomes obtained by MNBN given the list of the actual repository topics.

Table 1: Example of the MNBN outcomes for the *Spaceship-Generator* repository.

Actual Topics	Recommended topics
python, blender-scripts, spaceship, procedural-generation, game-development, 3d	shell, terminal, **3d**, opengl, **python**

Though MNBN works in practice, it suffers some limitations. First, the underlying model can recommend only featured topics that represent only a small set of all possible terms that can be potentially assigned to the analysed repository. For instance, *blender-scripts*[6] as well as *game-development* (which are not featured topics) could be recommended as possible topics because the project includes both *3d* and *python* topics. Thus, MNBN does not express all the concepts covered by a GitHub repository. As shown in the table, only two of the predicted topics match with the real ones. Moreover, in the case the repository already includes all suggested topics, MNBN is not able to recommend new ones. The second major limitation is the underlying structure needed for the training phase. MNBN requires a *balanced* dataset to deliver relevant items, i.e., each topic must have a similar number of README files. It is indeed challenging to satisfy such a constraint in practice, as topics are generally heterogeneous. Furthermore, repositories in GitHub are regularly updated with new topics, and thus, the training phase must take place several times to avoid outdated recommendations.

In recent years, many techniques have been conceived to predict users' interests by relying on the preferences collected from other users. Such techniques can be classified as *content-based* [**?**] where the relationships among items have been exploited to predict the most similar items, or *collaborative-filtering* [**?**] that calculates the missing 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. The former computes missing ratings by considering the ratings collected from similar users, whereas, the latter performs the same task by using the similarities among items [**?**].

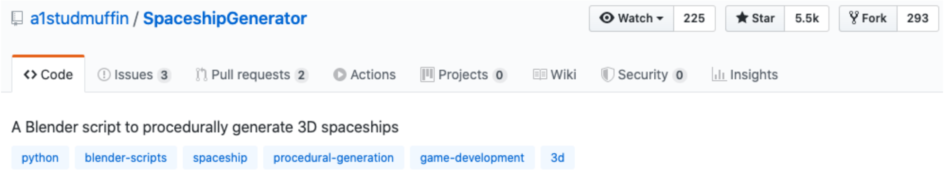
In the following section, we show that the proposed approach can recommend missing topics for GitHub repositories. Moreover, we also demonstrate that it is possible to increase the accuracy of MNBN by combining it with the proposed technique.

3 PROPOSED APPROACH

This section describes TopFilter, a recommender system that models the relationships among OSS projects using a graph representation, and exploits a collaborative filtering technique [**?**] to recommend relevant GitHub topics for the repository under development. Collaborative filtering techniques have been conceived in the e-commerce domain to recommend products [**?**], based on the assumption that *"if users agree about the quality or relevance of some items, then they will likely agree about other items"* [**?**]. TopFilter works following the same line of reasoning to mine GitHub topics: *"if projects have some tags in common, then they will probably contain other relevant tags"* [**?**].

In the following subsections, we describe two TopFilter configurations to recommend relevant topics by incorporating different types of input data. The first configuration exploits a collaborative-filtering technique, while the second one is a combination of both MNBN and TopFilter, aiming to improve the prediction performance of the original MNBN approach [**?**].

[4] https://github.com/topics
[5] https://github.com/a1studmuffin/SpaceshipGenerator
[6] blender is the most used python library to manipulate 3d objects. https://www.blender.org/


a1studmuffin / SpaceshipGenerator

A Blender script to procedurally generate 3D spaceships

Figure 1: A GitHub repository with different topics.

3.1 Recommending topics using a collaborative-filtering technique

Figure 2 depicts the architecture conceived to realize the TopFilter prototype. First, the `Data Encoder` component encodes GitHub repositories in a graph-based representation, and then `Similarity Calculator` computes similarities among all the managed projects. The `Recommendation Engine` component implements a *collaborative-filtering* technique to generate a ranked list of *top-N* topics which is eventually suggested to the developer to complement an initial list of topics given as input. We explain in detail the functionalities of each component as follows.

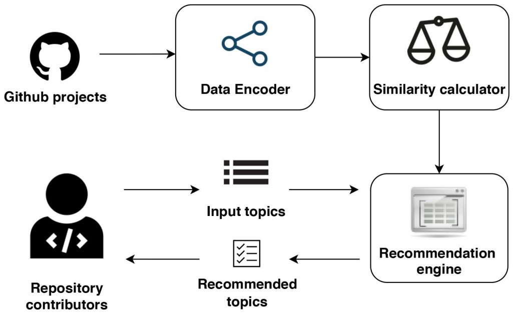


Figure 2: Overview of the TopFilter architecture.

3.1.1 Data Encoder. The mutual relationships among GitHub repositories and topics are encoded using a *project-topic matrix* [?]: Each row represents a project, and each column corresponds to a topic. In this sense, a cell in the matrix is set to 1 if the project in the corresponding row is tagged with the topic in the corresponding column, otherwise the cell is set to 0.

To build the project-topic matrix, raw topics are first pre-processed using various Natural Language Processing (NLP) techniques, such as stemming, lemmatization, and stop words removal. This aims to remove possible syntactical duplicated terms, e.g., *document* and *documents*, which are frequent in GitHub. Afterwards, the final matrix is constructed by means of the topics obtained through the pre-processing phase.

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 $L = \{t_1 = \mathit{machine\text{-}learning}; t_2 = \mathit{javascript}; t_3 = \mathit{database}; t_4 = \mathit{web}; t_5 = \mathit{algorithm}, t_6 = \mathit{algorithms}\}$. Moreover, the *project-topic inclusion* relationships is denoted as $\ni$. By parsing the projects, we discover the following inclusions: $p_1 \ni t_1, t_2, t_6; p_2 \ni t_1, t_3; p_3 \ni t_1, t_3, t_4, t_5; p_4 \ni t_1, t_2, t_4, t_5$. After the NLP normalization steps, the topics t_5 and t_6 collapse on the same term which is named as t_5. The final project-topic matrix is shown in Table 2.

Table 2: The *project-topic matrix* for the example.

	t_1	t_2	t_3	t_4	t_5
p_1	1	1	0	0	1
p_2	1	0	1	0	0
p_3	1	0	1	1	1
p_4	1	1	0	0	1

3.1.2 Similarity Calculator. This component relies on the previously encoded data to assess the similarity of given repositories. For explanatory reasons, we represent a set of projects and their topics in a graph, so as to calculate the similarities among the projects. For instance, Figure 3 depicts the graph-based representation of the project-topic matrix in Table 2. Two nodes in a graph are considered to be similar if they share the same neighbours by considering their edges. Such a technique has been successfully exploited by many studies to do the same task [?] in different domains.

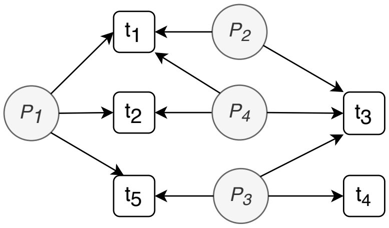


Figure 3: Graph representation for projects and topics.

Considering a project p that has a set of neighbor nodes $(t_1, t_2, .., t_l)$, the features of p are represented by a vector $\phi = (\phi_1, \phi_2, .., \phi_l)$, with ϕ_i being the weight of node t_i computed as the *term-frequency inverse document frequency* function as follows: $\phi_i = f_{t_i} \times log(|P| \times a_{t_i}^{-1})$, where f_{t_i} is the number of occurrences of t_i with respect to p, it can be either 0 and 1. $|P|$ is the total number of considered projects; a_{t_i} is the number of projects connecting to t_i via corresponding edges. Intuitively, the similarity between two projects p and q with their corresponding feature vectors $\phi = \{\phi_i\}_{i=1,..,l}$ and $\omega = \{\omega_j\}_{j=1,..,m}$ is computed as the cosine of the angle between the two vectors as given below.

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

where n is the cardinality of the union of topics by p and q.

3.1.3 Recommendation Engine. Given an input project p, and an initial set of related topics decided by the developer, the inclusion of additional topics can be predicted from the projects that are similar to p. In other words, TopFilter predicts topics' presence by means of those collected from the *top-k* similar projects using the following formula [?]:

$$r_{p,t} = \overline{r_p} + \frac{\sum_{q \in topsim(p)} (r_{q,t} - \overline{r_q}) \cdot sim(p,q)}{\sum_{q \in topsim(p)} sim(p,q)} \qquad (2)$$

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 1.

3.2 Combined use of MNBN and TopFilter

As already mentioned in Section 2, though MNBN works in practice, it suffers from some limitations. First, it can recommend only featured topics which account for a small fraction of all possible terms. Second, given that a repository already includes all suggested topics, MNBN is not able to recommend new ones. Moreover, the tool requires a *balanced* dataset, i.e., each topic must have a similar number of README files, and this is hard to come by in practice as topics are generally heterogeneous. With TopFilter, we attempt to improve MNBN by combining it with the collaborative-filtering technique presented in Section 3.1. The set of featured topics predicted by the MNBN model is used as input to feed TopFilter, which then runs the filtering process to deduce the inclusion of new topics.

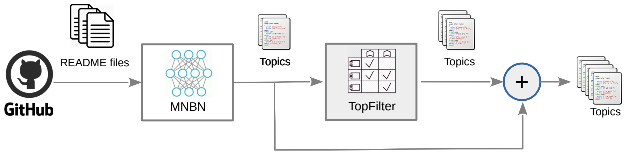


Figure 4: Overview of the combined approach.

Figure 4 depicts an overview of the combined use of MNBN and TopFilter: a list of featured topics computed by MNBN using README files is fed as input for TopFilter, which computes a list or ranked topics, including also non-featured ones. Finally, a list is generated by combining the topics computed by TopFilter with the top-N featured ones computed by MNBN, and recommended to developers.

We assume that TopFilter is beneficial in the following aspects:

- It is able to recommend non-featured topics, which are selected from similar repositories, independently from their nature;
- TopFilter recommends topics by iterating over refining steps: once we select some topics from the recommended ones, TopFilter can discover new topics using the selected ones as new input;
- MNBN does not provide additional recommendations given that the suggested topics are already included. As discussed in Section 5.1, TopFilter considerably improves the performance when more topics are incorporated as input.

In the following sections, we introduce the experiments conducted to evaluate the performance of TopFilter and its combined use with MNBN by means of different evaluation metrics.

4 EVALUATION MATERIALS AND METHODS

In this section, we report on the evaluation process conducted to study the performance of TopFilter. Section 4.1 presents the research questions we want to answer by means of the performed experiments. Section 4.2 gives an informative description of the datasets exploited in the evaluation. Section 4.3 and Section 4.4 describe the evaluation metrics and process, respectively. To facilitate future research, we made available the TopFilter tool together with the related data in a GitHub repository.[7]

4.1 Research Questions

The following research questions are addressed to study the performance of the proposed approach:

- *RQ_1: Which TopFilter configuration yields the best performance?* We investigate different configurations of TopFilter i.e., the number of input topics as well as the number of neighbour projects is varied, to find the best configuration.
- *RQ_2: To what extent can the accuracy of MNBN be improved by means of TopFilter?* We are interested in understanding if our proposed approach can be used to improve the accuracy of the original MNBN.

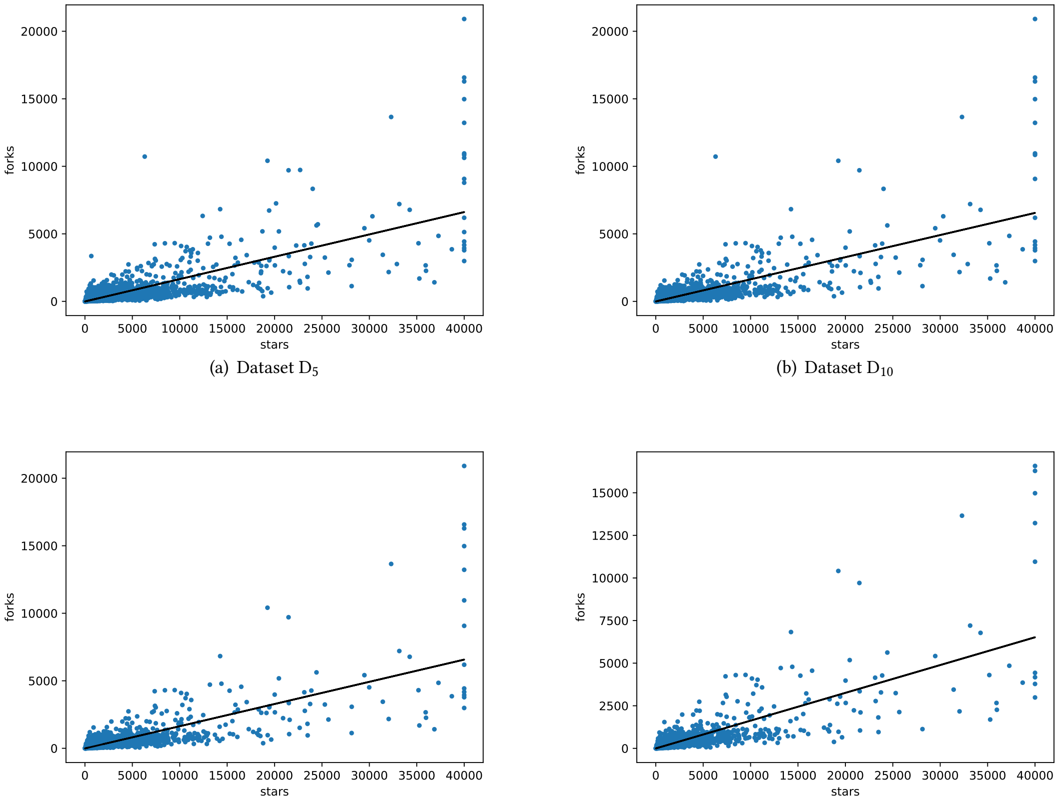
4.2 Data Extraction

To study our proposed approach, we reused the same dataset employed to evaluate MNBN which was already made available online in a replication package.[8] In particular, to investigate TopFilter's prediction performances, we populated five different datasets starting from the original MNBN corpus by varying the *cut-off* value t [?], i.e., the maximum frequency of the topic distribution (it is detailed in Section 4.4). In this way, we removed infrequent elements from the dataset to analyze their impacts on the overall recommendation phase. We firstly filtered the initial set of topics using their frequencies counted on the entire GitHub dataset. Afterwards, we removed irrelevant topics to reduce probable noise during the prediction phase.

We developed a filter by means of tailored Python scripts and applied it to the initial dataset. As a GitHub user can manually specify a topic list for a repository, many of them can contain infrequent or improper terms, i.e., the name of the author, duplicated values, or terms that rarely appear, to name a few. On one hand, imposing such a preprocessing phase reduces the number of repositories to analyze as well as topics to recommend. On the other hand, we improve the overall quality of recommendation by removing *"bad"* terms. This pruning phase can be done offline and does not affect the time required for the recommendation process.

Table 3 summarizes the main features of the datasets, i.e., D_1, D_5, D_{10}, D_{15}, and D_{20}, corresponding to different cut-off values t = $\{1, 5, 10, 15, 20\}$. *Avg. topics* is the average number of topics that the repositories include; *Avg. freq. topics* is the average frequency of the topics in the dataset.

[7] https://github.com/MDEGroup/TopFilter
[8] https://github.com/MDEGroup/MNB_TopicRecommendation/



(a) Dataset D_5

(b) Dataset D_{10}

(c) Dataset D_{15}

(d) Dataset D_{20}

Figure 5: A summary of the number of forks and stars for the datasets.

Table 3: Datasets.

	D_1	D_5	D_{10}	D_{15}	D_{20}
Number of repos	6,253	3,884	2,897	2,273	1,806
Number of topics	15,743	1,989	964	634	456
Avg. topics	9.9	8.4	8.0	7.8	7.7
Avg. freq. topics	3.9	16.5	24.1	28.1	30.5

As discussed in Section 5, removing infrequent topics improves the overall quality of the considered datasets: the collaborative filtering provides better prediction performance when there is enough data, i.e., topics in the training set. Once infrequent topics have been removed, all the repositories that contain less then five topics are filtered out from the dataset, as they contain little information to enable the collaborative filtering prediction. In particular, we remove around 2,300 repositories by increasing the cut-off value from 1 to 5. It means that the excluded repositories in Dataset D_5 are tagged with topics that rarely appear in the considered repositories. This finding is strengthened by the number of topics, which dramatically decreases to 1,989 when t=5. We stop at t=20 and consider Dataset D_{20} as the best one according to our metrics. Additionally, we observe that repositories are tagged by 9.9 and 7.7 topics on average for $t = 1$ and $t = 20$, respectively. This demonstrates that a large number of topics are not beneficial to the discoverability of a project.

From the list of projects under analysis, we exploited the GitHub API to retrieve their number of stars and forks. A summary of the retrieved data is plotted in Fig. 5. Forking is a means to contribute to the original repositories, furthermore, there is a strong correlation between forks and stars [?], as it can be further observed in Fig. 5. A project with a high number of forks means that it garners attention from the developers community. A repository with a large number of forks can be considered as a well-maintained and well-received project. Whereas, since commits also have an influence on source code [?], the number of commits is also a good indicator of how a project has been developed. As can be seen, most of the repositories possess a number of stars and forks less than 5,000. A small fraction of them have more than 30,000 forks and 10,000 stars. In this respect, we see that the datasets exhibit a wide variety of quality in terms of the number of forks and stars.

4.3 Metrics

In the scope of this paper, *success rate accuracy*, and *catalog coverage* are used to study the TopFilter performance as they have been

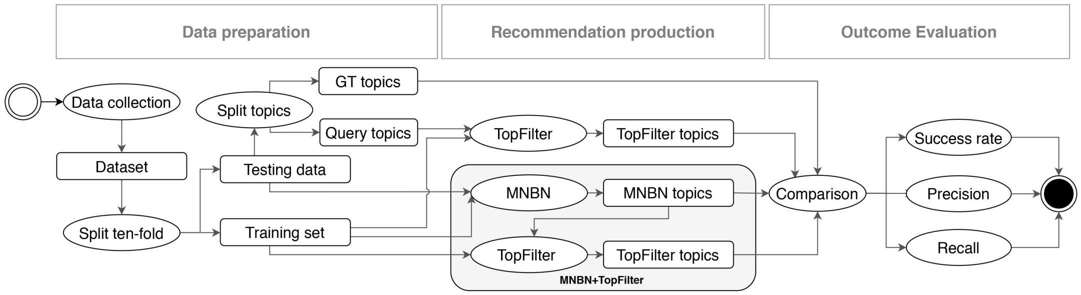


Figure 6: Evaluation Process.

widely exploited by related research [?]. First, we introduce the following notations as a base for further presentation:

- t is the frequency cut-off value of input topics, i.e., all topics that occur less than t times are removed from the dataset;
- τ is the number of topics that TopFilter takes as input;
- N is the cut-off value for the recommended ranked list of topic;
- k corresponds to the number of top-similar neighbor projects TopFilter considers to predict suggested topics;
- *GT(p)* is defined as a half of the extracted topics for a testing project p using as ground-truth data;
- $REC_N(p)$ is the top-N suggested topics sorted in a descending order;
- a recommended topic rt to a repository p is marked as a *match* if $rt \in REC(p)$;
- $match_N(p)$ is the set of items in $REC_N(p)$ that match with those in *GT(p)* for repository p.
- T is the set of all the available topics.

By means of the notations, the success rate, accuracy and coverage metrics are defined as follows.

Success rate@N. Given a set of testing projects P, success rate is defined as the ratio of queries that have at least a matched topic among the total number of queries.

$$success\ rate@N = \frac{count_{p \in P}(|match_N(p)| > 0)}{|P|} \tag{3}$$

Accuracy. Given a list of *top-N* libraries, *precision* and *recall* are utilized to measure the *accuracy* of the recommendation results. In particular, *precision* is the ratio of the *top-N* recommended topics found in the ground-truth data, whereas *recall* is the ratio of the ground-truth topics belonging to the N recommended items [?]:

$$precision@N = \frac{|match_N(p)|}{N} \tag{4}$$

$$recall@N = \frac{|match_N(p)|}{|GT(p)|} \tag{5}$$

Catalog coverage. Given the set of projects, we compare the number of recommended topics with the global number of the available ones. This metric measures the suitability of the delivered topics considering all the possible set of values.

$$coverage@N = \frac{\left|\cup_{p \in P} REC_N(p)\right|}{|T|} \tag{6}$$

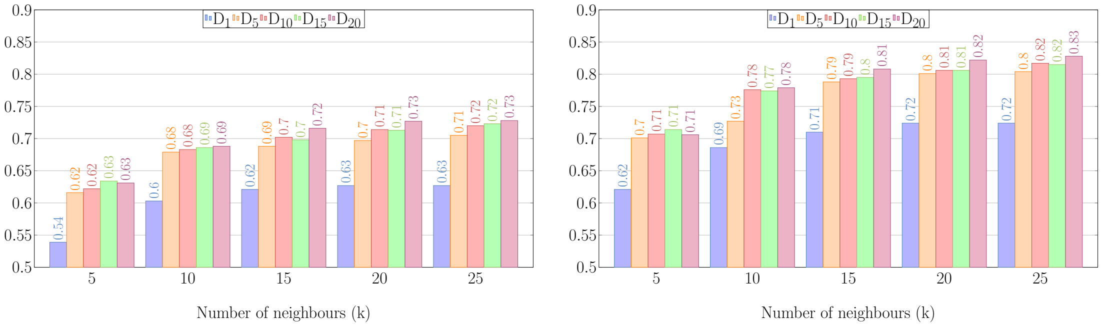
4.4 Evaluation process

The *ten-fold cross-validation* technique [?] has been used to assess the performance of TopFilter and its combined use with MNBN. Figure 6 depicts the evaluation process consisting of three consecutive steps, i.e., *Data Preparation*, *Recommendation Production*, and *Outcome Evaluation*, which going to be explained as follows.

Data Preparation. This phase is conducted to collect repositories that match the requirements defined in previous section from GitHub during the *Data collection* step. The dataset is then split into a training and testing set, i.e., `Split ten-fold`. Due to the different nature of the recommender systems (i.e., MNBN requires *README* files as input and training data, whereas TopFilter uses a set of assigned topics as input and for training) testing and training data needs to be specifically prepared for each approach. The `Split topic` activity resembles a real development process where a developer has already included some topics in the repository, i.e., *Query topics* and waits for recommendations.

Recommendation production. To enable the evaluation of TopFilter, we extracted a portion of topics from a given testing project, i.e., the ground-truth part. The remaining part is used as a query to produce recommendations. Since MNBN uses only README file(s) to predict a set of topics, it does not require any topics as input. The approach parses and encodes text files in vectors using the TF-IDF weighting scheme. Finally, the combined approach uses TopFilter as the recommendation engine which is fed by topics generated by MNBN. In this respect, both *Testing data* and *Training set* boxes are simplified to provide the needed data, i.e., README files and assigned topics to the recommender systems.

Outcome Evaluation. It is worth noting that we cannot directly compare TopFilter with MNBN since they rely on different input data. To be concrete, TopFilter is a supervised learning system that requires an initial set of assigned topics for the training, whereas MNBN is unsupervised learning and it uses only information mined from README files to recommend, without making use of any topics as input. Thus, we evaluate the performance of TopFilter by analyzing the recommendation results that are compared with those stored as ground-truth data to compute the quality metrics (i.e., *Success rate*, *Precision*, *Recall*, and *Catalog coverage*). The same metrics are used to evaluate the performance of the combined use of TopFilter and MNBN.



(a) N=5 (b) N=10

Figure 7: Success rate with 5 and 10 input topics.

5 EXPERIMENTAL RESULTS

This section analyzes the performance of TopFilter as well as the combination of MNBN with TopFilter by addressing the two research questions in Section 5.1 and Section 5.2, respectively.

5.1 *RQ$_1$: Which TopFilter configuration yields the best performance?*

As presented in Section 4, given a testing project p, a certain number of topics is used as input, i.e., τ, and the remaining ones are saved as ground truth data, i.e., *GT(p)*. To find the configuration obtaining the best prediction performances, we experimented by varying the available parameters, i.e., the number of top-similar neighbour projects k, the number of recommended items N. To be more concrete, we chose two different values of N, i.e., $N = \{5, 10\}$, and five values of k, i.e., $k = \{5, 10, 15, 20, 25\}$. Furthermore, we also considered all the five datasets defined in Section 4.2, i.e., D_1, D_5, D_{10}, D_{15}, and D_{20}. The value of τ is always considered as half of the number of topics already assigned to the project under analysis. In particular, given a testing project, the first half of the related topics are used to construct the query. The average success rates obtained by running the ten-fold cross-validation technique with TopFilter are depicted in Fig. 7(a) and Fig. 7(b).

Overall, it is evident that infrequent topics negatively affect the prediction outcomes. With D_1, i.e., all projects are considered, TopFilter obtains a low success rate by both configurations N=5 and N=10, compared to the results of other cut-off values t. For instance, with k=5, TopFilter gets 0.54 as success rate for D_1, meanwhile it gets 0.62 as success rate for D_5 and D_{10}, and 0.63 for D_{15} and D_{20}. The same trend can also be witnessed with other values of k, i.e., $k = \{10, 15, 20, 25\}$: TopFilter achieves a better performance when we consider a dataset with more topics for each project. This is understandable since TopFilter relies on the availability of training data to function: the more topics it has, the better it can compute the similarities among projects (cf. Fig. 3 and Eq. 1), and thus it is able to find more relevant topics (cf. Eq. 2). Similarly, for N=10, i.e., we consider a longer list of recommendations, TopFilter obtains a better success rate when more detailed training data is exploited.

Next, we examine the influence of the number of neighbors k on the prediction performance. For N=5, from Fig. 7(a) it is clear that incorporating more neighbor projects for recommendation helps improve the performance considerably. Take as an example, for k=10 the best success rate is 0.69 obtained for D_{20}, and the corresponding score achieved with k=25 is 0.73 for D_{20}. For N=10, we see a clear gain in performance when more neighbors are considered for the computation of recommendations. The best obtained success rate is 0.83 when k=25 for D_{20}. As a whole, we conclude that increasing the number of neighbors used for computing missing ratings in the project-topic matrix is beneficial to the final recommendations. On the other hand, this also increases the computational complexity as comprehended in Eq. 2. Therefore, it is necessary to maintain a trade-off between accuracy and efficiency when deploying TopFilter by choosing a suitable value of k.

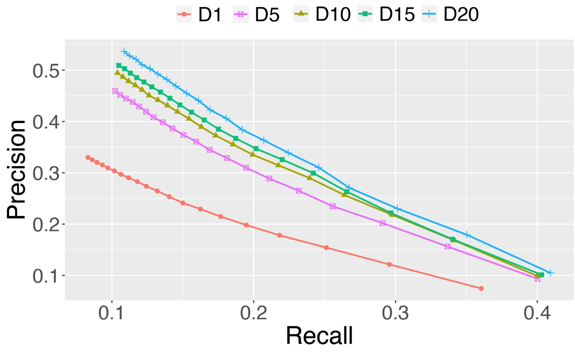


Figure 8: Precision/recall curves.

To further study TopFilter's performance, we computed and depicted in Fig. 8 the precision/recall curves (PRCs) for all the considered datasets. For this setting, the number of recommended items N was varied from 1 to 20, aiming to study the performance for a long recommendation list. Each dot in a curve corresponds to precision and recall obtained for a specific value of N. Furthermore, we fixed k=25 since this number of neighbors brings the best prediction outcomes among others, while it still maintains a reasonable execution time. As a PRC close to the upper right corner represents a better precision/call [**?**], Fig. 8 demonstrates that by considering a dataset with more topics for each repository, TopFilter yields a better prediction performance. In particular, the worst precision/recall relationship is seen by D_1, while the best one is obtained by D_{20}. Overall, these results are consistent with those presented in

Fig. 7(a) and Fig. 7(b): using projects consisting of more input topics helps TopFilter enhance its performance substantially.

We investigate if TopFilter can recommend a wide range of topics to repositories by considering catalog coverage. The metric measures the percentage of recommended topic in the training data that the model is able to suggest to a test set, and a higher value corresponds to a better coverage. Table 4 reports the average coverage value for the datasets, i.e., D_1, D_5, D_{10}, D_{15}, D_{20}.

Table 4: Catalog Coverage.

N	$\mathbf{D_1}$	$\mathbf{D_5}$	$\mathbf{D_{10}}$	$\mathbf{D_{15}}$	$\mathbf{D_{20}}$
2	2.313	1.433	1.075	0.886	0.715
4	3.925	2.362	1.753	1.440	1.143
6	5.494	3.232	2.346	1.858	1.478
8	7.075	4.035	2.835	2.185	1.737
10	8.720	4.788	3.239	2.458	1.920
12	10.385	5.472	3.615	2.702	2.082
14	12.073	6.120	3.934	2.915	2.223
16	13.872	6.729	4.216	3.088	2.339
18	15.753	7.252	4.475	3.244	2.442
20	17.746	7.770	4.699	3.369	2.521

From the table, we see that by considering a longer list of items, i.e., increasing N, a better coverage is gained. Furthermore, using a higher cut-off value t has a negative impact on the global catalog coverage. For instance, with D_1 and N=2, we obtain 2.313 as coverage, and this score gradually decreases along t, and shrinks to 0.715 with D_{20}. Similarly, by other values of N, catalog coverage is large for a low t and small for a high t. This can be explained as follows: setting a high value of t means that a large amount of training data is discarded, and thus removing also topics. Altogether, we see that using a denser dataset for training, i.e., projects with more topics, is beneficial to success rate, accuracy, but not to catalog coverage.

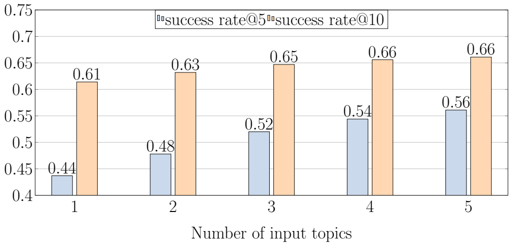


Figure 9: Success rates for τ={1, 2, 3, 4, 5} on D_{20}.

We are also interested to understanding how τ impacts on the prediction performance by performing a cross validation with τ={1, 2, 3, 4, 5}. Moreover, to simplify the evaluation, we fixed the number neighbors k to 25 and the number of topics t to 20 and applied TopFilter on the dataset D_{20}. Figure 9 reports the average success rate obtained for two values of N, i.e., N={5, 10}. The figure demonstrates an evident outcome: by using more input topics as input data, we get a better prediction performance. Take as an example, for τ=1, we get a success rate of 0.44 and 0.61 for N=5 and N=10, respectively. When we use 5 topics to feed TopFilter, the corresponding success rate is 0.56 and 0.66 for N=5 and N=10, respectively. This happens due to the fact that by considering more input topics, TopFilter is able to better determine the similarity between the testing project and those in the training data, as shown in Eq. 2, which eventually allows it to mine more relevant topics.

Answer to RQ$_1$. Given a project, TopFilter achieves a better success rate and accuracy when more similar projects are considered for recommendation. A higher cut-off value t negatively impacts on the coverage; and using more topics as input data helps the system improve its performance.

5.2 *RQ$_2$: To what extent can the accuracy of MNBN be improved by means of TopFilter?*

As already reasoned in Section 4.4, it is not feasible to directly compare TopFilter with MNBN, as they are based on different recommendation mechanisms. TopFilter relies on a supervised learning technique, requiring an initial set of assigned topics for the training. Meanwhile, MNBN works on the basis of an unsupervised learning system, which needs only data mined from README files to recommend, without being fed with any input topics.

In this research question, we aim to show that combining MNBN with TopFilter (referred with *MNBN+TopFilter* hereafter) helps boost up the recommendation outcomes provided by MNBN. In particular, we run TopFilter on the outputs produced by MNBN, attempting to generate a more relevant list of items.

We conducted experiments on two of the selected datasets, i.e., D_1 and D_{20}, as they correspond to two distinct levels of data completeness: D_1 has more projects but with a lower quality, while D_{20} has fewer data but with a higher quality. In the experiment, τ was set to 5 since through RQ$_1$, we realized that this value fosters the best performance, among others (see Fig. 9). The success rate, precision, recall and coverage scores obtained for D_1 and D_{20} are reported in Table 5 and Table 6, respectively.

The results in Table 5 demonstrate that compared to TopFilter, MNBN obtains a better prediction performance in terms of success rate, recall and precision for a ranked list with a low number of items, i.e., N ≤ 4. In particular, with N=2, MNBN gets 0.240 as success rate, while the corresponding score by *MNBN+TopFilter* is 0.148. The same trend can be seen with recall and precision. This is understandable as MNBN relies on only README file(s) to function, and its performance is not affected by the number of considered topics. In contrast, TopFilter needs input topics to provide recommendation, that is the reason why for a dataset with a low quality dataset (with respect to the number of topics), TopFilter gets a moderate performance.

Table 5 also shows that the combined use of MNBN and TopFilter outperforms MNBN when we consider a longer list of recommended items. To be concrete, starting from N=6 (i.e., the row marked with gray), all the metrics computed with *MNBN+TopFilter* are superior to those computed with MNBN. For example, *MNBN+TopFilter* gets 0.675 as the maximum success rate, while the corresponding score by MNBN is 0.662. More importantly, *MNBN+TopFilter* always achieves a much better coverage than that of MNBN: By the best configuration, *MNBN+TopFilter* reaches a coverage of 3.544, which is much higher than 0.768 obtained by MNBN. In other words, our proposed approach is able to recommend a wider range of topics than MNBN. This can be explained by the fact that TopFilter takes

Table 5: Comparison between MNBN and *MNBN+TopFilter* using Dataset D_1.

	Success rate		Recall		Precision		Catalog coverage	
N	MNBN	*MNBN+TopFilter*	MNBN	*MNBN+TopFilter*	MNBN	*MNBN+TopFilter*	MNBN	*MNBN+TopFilter*
2	0.240	0.148	0.026	0.017	0.137	0.077	0.175	0.411
4	0.383	0.287	0.042	0.035	0.123	0.081	0.314	0.617
6	0.441	0.449	0.049	0.064	0.104	0.098	0.398	0.919
8	0.499	0.531	0.055	0.085	0.093	0.098	0.474	1.272
10	0.550	0.572	0.061	0.100	0.087	0.092	0.557	1.626
12	0.584	0.603	0.066	0.111	0.081	0.086	0.620	2.008
14	0.601	0.624	0.067	0.120	0.074	0.080	0.658	2.431
16	0.616	0.643	0.068	0.128	0.067	0.074	0.687	2.825
18	0.632	0.660	0.069	0.136	0.062	0.070	0.718	3.169
20	0.662	0.675	0.073	0.143	0.060	0.066	0.768	3.544

Table 6: Comparison between MNBN and *MNBN+TopFilter* using Dataset D_{20}.

	Success rate		Recall		Precision		Catalog coverage	
N	MNBN	*MNBN+TopFilter*	MNBN	*MNBN+TopFilter*	MNBN	*MNBN+TopFilter*	MNBN	*MNBN+TopFilter*
2	0.363	0.217	0.035	0.031	0.206	0.118	0.263	0.249
4	0.600	0.389	0.075	0.063	0.221	0.119	0.562	0.444
6	0.635	0.601	0.094	0.121	0.187	0.153	0.715	0.612
8	0.680	0.704	0.106	0.171	0.159	0.162	0.810	0.848
10	0.701	0.754	0.116	0.204	0.140	0.156	0.890	1.022
12	0.719	0.788	0.124	0.230	0.124	0.146	0.950	1.178
14	0.733	0.808	0.130	0.254	0.111	0.138	0.994	1.330
16	0.745	0.829	0.135	0.274	0.101	0.131	1.035	1.463
18	0.759	0.840	0.143	0.290	0.095	0.123	1.090	1.582
20	0.772	0.855	0.150	0.306	0.090	0.117	1.148	1.701

into consideration a set of topics as input, and the more data it has, the larger the set of topics it can recommend.

By examining Table 6, we encounter a similar outcome compared to that of Table 5: *MNBN+TopFilter* outperforms MNBN by all the quality metrics, i.e., success rate, recall, precision and coverage. Especially, using D_{20} as the training data, *MNBN+TopFilter* improves the overall success rate considerably, with respect to using D_1. This further confirms our findings in RQ_1: A denser dataset facilitates the capability of recommending a more relevant set of topics.

Answer to RQ_2. Compared to MNBN, the combined used of MNBN and TopFilter substantially improves the prediction performance with respect to success rate, precision and recall. While MNBN suffers a low catalog coverage, MNBN+TopFilter is able to recommend a wide range of topics to repositories.

6 THREATS TO VALIDITY

This section discusses the threats that may affect the results of the evaluation. We also list the countermeasures taken to minimize these issues.

The *internal validity* could be compromised by the dataset features, i.e., the number of projects for each topic, the number of results. We tackle this issue by varying the aforementioned parameters to build datasets with different characteristics. In this way, several settings have been used to evaluate TopFilter's overall performances.

External validity concerns the rationale behind the selection of the GitHub repositories used in the assessment. As stated in Section 4, we randomly downloaded repositories by imposing a quality filter on the number stars. Nevertheless, some repositories could be tagged with topics that can affect the quality of the graph computed in the data extraction phase. To be concrete, a user can label a repository using terms that are not descriptive enough, i.e., using infrequent or duplicated terms in the topic list. To deal with this issue, we applied the topic filter as described in Section 4.2 to reduce any possible noise during the graph construction phase.

Threats to *construction validity* are related to the choice of MNBN as the baseline in the conducted experiments. First, the availability of the replication package allows us to perform a comprehensive evaluation. As we claimed before, the two approaches are strongly different from the construction point of view including the recommendation engine and data extraction components. To make the comparison as fair as possible, we ran MNBN on the same datasets by adapting the overall structure for the ten-fold cross-validation evaluations.

7 RELATED WORK

Immediately after the introduction of topics in the GitHub platform, the Repo-Topix tool was presented [?]. Such a tool relies on parsing the README files and the textual content of a given GitHub repository to suggest topics automatically. As a first step, the tool applies standard NLP techniques on the input artifacts. Then, it filters an initial set of topics by exploiting the TF-IDF scheme and a regression model to exclude *"bad"* topics. As the final step, Repo-Topix computes a custom version of the Jaccard distance to discover additional similar topics. A rough evaluation based on the n-gram ROUGE-1 metrics has been conducted by counting the number of overlapping units between the recommended topics and the repository description. Nevertheless, a replication package with the complete dataset and the source code of the tool is not available, and this hampers further investigations and comparisons.

A collaborative topic regression (CTR) model has been proposed to extract topics from a given GitHub repository [?]. The final aim is to recommend other similar projects given the input one. For a pair of user-repository, the approach uses a Gaussian model to compute matrix factorization and extract the latent vectors given a pre-computed matrix rating. Additionally, a probabilistic topic modeling is applied to find topics from the repositories by analyzing high frequent terms. The approach was evaluated by conducting five-fold cross-validation on a dataset composed of 120,867 repositories. Such evaluation considers user-repository pairs that have at least 3 watches. Differently from TopFilter, this approach can recommend GitHub repositories that are relevant with respect to the topics of the input repository.

Lia *et al.* [?] propose a user-oriented portrait model to recommend a set of GitHub projects that can be of interest for a given user. An initial set of labels is obtained by running the LDA algorithm on the textual elements of a repository, i.e., issues, commits, and pull requests. Then, the approach exploits a project familiarity technique that relies on the user's behavior, considering the different repositories operation. Such a strategy enables the collaborative filtering technique that exploits two kinds of similarity, i.e., attribute and social similarity. The former takes into account personal user information such as company, geographical information and the time when the account has been created. The latter computes similarity scores considering the proportion of items contributed by the user. The approach was evaluated by considering 80 users with an average of 1,894 different behaviors for each one. By considering the first two months of activity in 2016 as a test set, the assessment shows that the approach improves the performances in terms of precision, recall, and success rate. Though both TopFilter and the presented work [?] make use of collaborative-filtering techniques, the former is designed to recommend topics to be assigned to an input GitHub repository, whereas the latter recommends GitHub repositories that can be of interest for a given user.

A model-based fuzzy C-means for collaborative filtering (MFCCF) has been proposed [?] to recommend relevant human resources during the GitHub project development. Similarly to our approach, the proposed model encodes relevant information about repositories in a graph structure and extracts from it a sparse test sub-graph. This is a preparatory phase to enable the fuzzy C-means clustering technique. Using the computed sparse sub-graph as the centre of the cluster, the model can handle the sparsity issue that generally arises in the CF domain. Then, MFCCF computes the Pearson Correlation for each pair user-item belonging to a cluster and retrieves the top-N results. The evaluation was performed using the GHTorrent dump to collect the necessary information. Using ten projects as the testing dataset, the results of MFCCF are compared with the ones chosen by the human resource department of the considered company. The results demonstrate the effectiveness of the approach with an accuracy of 80% on average.

The REPERSP tool [?] recommends GitHub projects by exploiting users' behaviour. As the first step, the tool computes the similarities between projects using the TF-IDF weighting scheme to obtain the content similarity matrix. Additionally, REPERSP captures the developer's behaviour by considering her activity on GitHub, i.e., create, star, and fork actions over projects. A different value is assigned for each type of action to create a user-project matrix. Finally, the tool combines the two similarity matrices to deliver the recommended projects. To assess the quality of the work, REPERSP was compared with the traditional collaborative filtering techniques, i.e., user-based and item-based. The study was conducted over two groups with different users, projects, and purposes. The results show that the proposed tool outperforms the considered baseline in terms of accuracy, precision, and recall.

Besides GitHub projects, tags and topics are successfully used in different contexts, i.e., in social networks. Purushotham *et al.* [?] propose a hierarchical Bayesian model that relies on a topic model to provide final users with relevant items. By combining LDA and matrix factorization techniques, the proposed model is able to reduce the sparsity problem that typically occurs when the collaborative filtering is employed. After this preprocessing phase, the hierarchical Bayesian model is tuned with several parameters to maximize the prediction performances. The models were evaluated by using two large real-world datasets tailored for music and bookmark recommendations. The experimental results show that the proposed model outperforms the classical CTR model with respect to recall.

8 CONCLUSIONS AND FUTURE WORK

GitHub is nowadays among the most popular platforms to handle and maintain OSS projects. Topics have been introduced in 2017 to promote projects' visibility on the platform. In this work, we presented TopFilter, a collaborative filtering-based recommender system to suggest GitHub topics. By encoding repositories and related topics in a graph-based representation, we built a project-topic matrix and applied a syntactic-based similarity function to predict missing topics. To assess the prediction performances, we combined our approach with a well-founded system based on an ML algorithm, i.e., MNBN. In particular, by taking as input in TopFilter, the output produced by MNBN, we showed that by means of a combined usage of such approaches, it is possible to obtain a significant boost in topic predictions. Nevertheless, the accuracy did not reach higher values in all the experimental settings. To our best knowledge, this depends very much on the similarity function used in the recommendation engine as well as on the heterogeneity of the dataset. Thus, we plan to extend TopFilter by adding different degrees of similarity, e.g., semantic analysis on topics. Moreover, we can enlarge the evaluation by considering other common metrics in the collaborative filtering domain such as sales diversity and novelty. All of this is planned as future work.

ACKNOWLEDGEMENTS

The research described in this paper has been carried out as part of the CROSSMINER Project, which has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant 732223.