Applied Intelligence (2023) 53:9708–9730
https://doi.org/10.1007/s10489-022-03864-y

HybridRec: A recommender system for tagging GitHub repositories

Juri Di Rocco[1] · Davide Di Ruscio[1] · Claudio Di Sipio[1] · Phuong T. Nguyen[1] · Riccardo Rubei[1]

Accepted: 7 June 2022 / Published online: 11 August 2022

Abstract

Software repositories are increasingly essential to support the management of typical artifacts building up projects, including source code, documentation, and bug reports. GitHub is at the forefront of this kind of platforms, providing developer with a reservoir of code contained in more than 28M repositories. To help developers find the right artifacts, GitHub uses topics, which are short texts assigned to the stored artifacts. However, assigning inappropriate topics to a repository might hamper its popularity and reachability. In our previous work, we implemented MNBN and TopFilter to recommend GitHub topics. MNBN exploits a stochastic network to predict topics, while TopFilter relies on a syntactic-based function to recommend topics. In this paper, we extend our work by building HybridRec, a recommender system based on stochastic and collaborative-filtering techniques to generate more relevant topics. To deal with unbalanced datasets, we employ a Complement Naïve Bayesian Network (CNBN). Furthermore, we apply a preprocessing phase to clean and refine the input data before feeding the recommendation engine. An empirical evaluation demonstrates that HybridRec outperforms three state-of-the-art baselines, obtaining a better performance with respect to various metrics. We conclude that the conceived framework can be used to help developers increase their projects' visibility.

Keywords Recommender systems · GitHub tagging · Mining software repositories · Collaborative filtering · Bayesian network

1 Introduction

Over the last decade, open-source software repositories have gained a prominent role in storing and managing software projects. Different kinds of artifacts, including source code, mailing lists, bug tracking systems, and documentation, are stored and managed homogeneously employing powerful technologies. In such a context, GitHub is playing the role of forefront platform managing more than 190M repositories, with more than 28M of them being available to the public.[1] The platform offers developers the possibility to classify the stored artifacts by means of topics, i.e., it introduced the possibility of labeling the stored repositories only in 2017 to help developers increase the reachability of their repositories. The assigned labels allow users to characterize projects, e.g., with respect to the provided functionalities and employed technologies.

However, assigning wrong labels to a given repository can compromise its popularity [3], and even worse, prevent developers from finding projects that they might be willing to contribute. In this respect, we come across the following motivating question:

> "*Which label should I use to annotate this new project managed by means of the employed OSS repository?*"

Repo-topix [10] is a mechanism based on information retrieval techniques, and it has been developed and integrated into GitHub to recommend topics. In an attempt to improve repo-topix, e.g., in terms of the variety of the suggested topics, we proposed MNBN [8] as a novel approach based on a Multinomial Naïve Bayesian network to recommend relevant topics starting from the textual content,

✉ Davide Di Ruscio
davide.dirsucio@univaq.it

Juri Di Rocco
juri.dirocco@univaq.it

Claudio Di Sipio
claudio.disipio@graduate.univaq.it

Phuong T. Nguyen
phuong.nguyen@univaq.it

Riccardo Rubei
riccardo.rubei@graduate.univaq.it

[1] Department of Information Engineering, Computer Science and Mathematics, Università degli studi dell'Aquila, L'Aquilla, Italy

[1] The numbers are collected at the time of writing.

i.e., README files, 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.[2] Subsequently, we successfully developed another approach, called TopFilter [7], that relies on a collaborative filtering technique to extend the recommendation capabilities of MNBN, taking into consideration non-featured topics. Given an initial set of topics already assigned to the GitHub repository of interest, a project-topic matrix is created starting from a graph-based format that encodes the reciprocal relationships between projects and relationships. Then, the underpinning recommendation algorithm relies on a syntactic similarity function based on featured vectors to recommend the most similar topics.

In our recent work [7], we show that the combined use of MNBN and TopFilter (named TopFilter$^+$ hereafter[3]) can improve the results obtained by MNBN. However, according to further investigations, there is still room for improvement for TopFilter$^+$. In particular, it is necessary to deal with unbalanced datasets (as typically occur in real contexts) as well as to enhance the quality of the recommended items. In this work, we further advance the existing tools by proposing a hybrid recommender system, named HybridRec, which retrieves topics for repositories using a combination of stochastic and collaborative filtering recommendation strategies (thereby yielding the name *hybrid*), being capable of dealing with unbalanced and heterogeneous datasets. To enable both techniques we select the most used artifacts[4] by relying on popular topics. A crucial step lies in the preprocessing phase of the dataset and topics, which dramatically differs from the one set in place for the GitHub repositories. In particular, the type of available metadata for each project can affect the recommendation outcomes as well as the efficiency of the underpinning techniques.

To evaluate HybridRec, we perform a series of experiments, exploiting real-world datasets collected from GitHub. Moreover, we compare HybridRec with MNBN [8], TopFilter, and TopFilter$^+$ [7], which can be considered as among state-of-the-art approaches to the recommendation of topics for GitHub repositories. The experimental results show that HybridRec performs better than the baselines, recommending highly relevant topics in most cases. In this sense, the contributions of this paper are summarized as follows:

- A comprehensive discussion on open challenges in retrieving relevant information from the GitHub ecosystem;
- A hybrid recommender system, called HybridRec, built on top of *(i)* a Complement Naïve Bayesian Network [23], *(ii)* a collaborative-filtering technique, and *(iii)* a rule-based preprocessing phase to suggest topics for GitHub repositories;
- An empirical study using real-world datasets collected from GitHub and MVN Repository, comparing HybridRec with three state-of-the-art baselines, i.e., MNBN [8], TopFilter, and TopFilter$^+$ [7];
- The tools and datasets developed and curated through our work have been published to foster future research.[5]

The paper is structured into the following sections. In Section 2, we motivate the problem addressed in this paper and the research objective we aimed to achieve. The conceived approach is presented and evaluated in Section 3 and Section 4, respectively. Afterward, the obtained results are quantitatively and qualitatively analyzed in Section 5. Next, we discuss the probable threats to the validity of the performed experiments in Section 6, while the related work is reviewed in Section 7. Finally, we draw some perspective work and conclude the paper in Section 8.

2 Background and motivation

This section provides an introduction to the importance of topics in GitHub. We identify several challenges that arise during the mining process of the ecosystem. The research problem addressed in this paper is then introduced.

2.1 Problem statement

Software developers use open-source software (OSS) repositories to publish and disseminate their work. GitHub is amongst the most popular platforms that offer open environments where developers can share their code and interact with each other. To assist developers in approaching projects of their interest, GitHub provides users with tools and techniques that help narrow down the search scope and increase the visibility of its projects. In particular, to characterize the stored artifacts, GitHub leverages *featured topics*,[6] i.e., a list of the most popular and active topics, which are periodically monitored and updated. Such a public list indicates the community's trend in terms of the most used topics. Developers have manually assigned

[2] https://github.com/topics

[3] Originally, there was no specific name for the combined use of MNBN and TopFilter. In the scope of this paper, we name the combination as TopFilter$^+$ to facilitate the presentation.

[4] For the sake of the presentation, the terms "*project*," "*repository*," and "*artifact*" are used interchangeably throughout the paper.

[5] https://github.com/MDEGroup/HybridRec

[6] https://github.com/topics

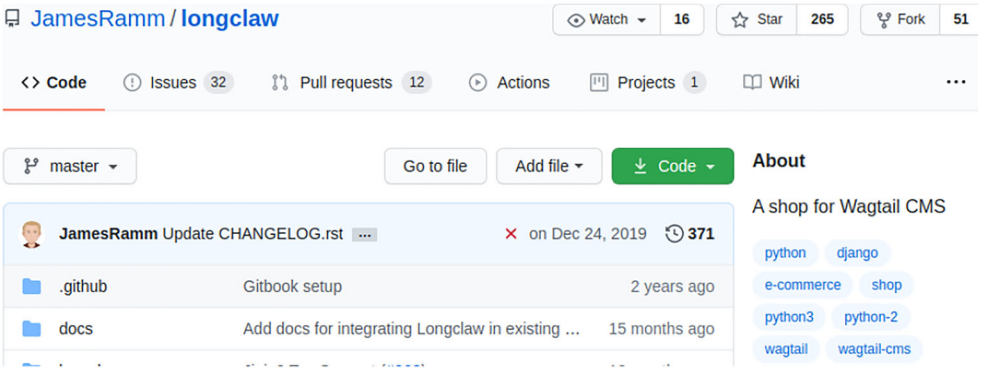

Fig. 1 The *longclaw* repository from GitHub with different topics

GitHub topics according to their experience as well as to the content of the repositories at hand. Nevertheless, topics assigned in such a way might be inaccurate or represent wrong concepts, thus compromising the visibilities of projects.

Figure 1 represents an explanatory example consisting of the *longclaw* repository,[7] which implements extensions of Wagtail CMS[8] to enable the development of typical *e-commerce* functionalities. By looking at the list of topics, users can easily understand that the project employs *django*, a *python* web framework. Though the given topics characterize the repository's features, some of them might be considered redundant, e.g., *python3*, *python-2*, and *wagtail-cms*. Moreover, only three topics are considered to be *featured* as they are given in Table 1.

The example in Fig. 1 shows that on the one hand, user-specified topics are usually more extensive than the *featured* ones. On the other hand, two different topics can express the same concept, and a larger set of topics could affect the prediction accuracy of automatic approaches that might rely on such data.

Summary. Topics are an effective means of summarizing the main features of a GitHub repository. Furthermore, the proper use of topics facilitates the searchability and discoverability of different items. Thus, there is the need for techniques and tools to automatically generate topics to prevent them from being misleadingly/wrongly established and not correctly reflecting the contents of the considered projects.

2.2 Challenges in mining the GitHub ecosystem

According to existing work [5, 15], extracting valuable data from GitHub is a daunting task and exhibits several pitfalls. We identify the following challenges to be undertaken to conceive the desired recommender system:

C1: *Data redundancy*. GitHub topics are specified (or even created) by users to classify their repositories. However, this manual process results in inaccurate labeling in some situations. For instance, a user can define both *python* and *python-3* as topics for its repository, which are redundant. Refining the list of topics by removing such kinds of topics can improve the discoverability of the repository;

C2: *Structure of available metadata*. Concerning the available sources of knowledge, a standard GitHub project is described by one or more README file(s), a brief description, and possibly by a Wiki. Furthermore, there are different kinds of accessible metadata, e.g., commits, issues, forks, and stars. Though GitHub provides publicly access in most of the cases, extracting valuable information from the data mentioned above requires a set of tailored preprocessing techniques;

C3: *Popularity mechanisms*. GitHub provides users with the *star* and *forking* mechanisms to assess the popularity of a given repository [3, 14]. The former is used to improve the visibility of a project. The latter is typically employed when a developer wants to "*freely experiment with changes without affecting the original project*."[9] Moreover, GitHub groups the most popular projects under a curated list, i.e., featured topics. In such a way, the popularity of a repository helps the mining process filter out unuseful artifacts, e.g., toy and dummy projects. Though many software artifact repositories provide popularity mechanisms, there is no uniform way to define the popularity of an artifact. For instance, GitHub includes many elements, i.e., *star*, *forking*, number of committers, etc., to assess the repository popularity, while Maven defines the popularity of an artifact by relying on the number of usages, i.e., when another project employs the artifact;

[7] https://github.com/JamesRamm/longclaw
[8] https://github.com/wagtail/wagtail
[9] https://docs.github.com/en/free-pro-team@latest/github/getting-started-with-github/fork-a-repo

Table 1 The *longclaw* repository topics

Topics	GitHub featured topics
python, django, e-commerce, shop, python3, python-2, wagtail, wagtail-cms	python, django, wagtail

C4: *Crawling and Data Dump*. The availability of input data is a crucial aspect of the recommendation process. To gather them from a platform that stores OSS projects we can *(i)* use a crawler or *(ii)* rely on a data dump stored in some database format. Concerning the crawling activity, GitHub exposes its API to obtain all needed data by exploiting different libraries, e.g., JGit,[10] PyGitHub,[11] to name a few. Furthermore, GHTorrent [11] offers a regularly updated data dump of the entire platform in several formats, e.g., SQL, and MongoDB. It is worth noting that GHTorrent does not include the actual content of each repository, but only stores metadata;

C5: *Configurations of the underpinning recommender systems*. Depending on the features of the considered OSS ecosystem, the employed recommendation algorithm must be adapted accordingly by changing its internal configurations. Such a phase can rely on different parameters that vary according to the nature of the used algorithms.

Research objective. Considering the need mentioned above and the corresponding challenges, we propose a workable solution to the recommendation of topics to GitHub repositories. We conceptualize a hybrid use of a complement naïve bayesian network and a collaborative-filtering technique.

3 Proposed approach

In this section, we present in detail HybridRec, the proposed recommender system to provide topics for GitHub repositories. HybridRec parses textual contents, e.g., README files, Wiki contents, and commit messages collected from GitHub, and employs a Complement Naïve Bayesian Network [23], or CNBN for short, to extract preliminary topics. The predicted topics are then fed as input to retrieve additional relevant topics by means of a collaborative-filtering technique. The outcome of this phase is combined with the preliminary topics to yield the final recommendations.

Figure 2 illustrates the architecture of HybridRec, which consists of two main components, namely *(i)* **ST**: Probability based recommendation using a stochastic network [23]; and *(ii)* **CF**: Collaborative-filtering based recommendation [21, 27]. The following subsections describe these components in greater detail.

3.1 ST: The stochastic-based component

The architecture of the first component is depicted in Fig. 3. Data is crawled from GitHub and then vectorized by the *TF-IDF encoding* component. The obtained data is used to feed the CNBN component, which returns a set of most probable topics. By resembling the choice made in the original work [8], we consider only the *featured* topics, i.e., the most popular according to GitHub's statistics. In such a way, we can train CNBN with the projects labeled with featured topics that are representative in terms of coverage. Furthermore, various NLP techniques are applied to both predicted and actual topics to improve the quality of the outcomes, as we explain as follows.

3.1.1 TF-IDF encoding

Given a repository, this module vectorizes the textual content of its artifact descriptions, using the *scikit-learn* library[12] that provides all the functionalities to encode textual content by finding the most representative terms. `Encoder` computes the inverse document-frequency using the following formula:

$$idf(t) = \log \frac{1+n}{1+df(t)} + 1 \quad (1)$$

where n is the total number of documents in the document set; $df(t)$ is the number of documents in the document set that contain term t. A previous study [16] showed that applying such a weighting scheme should possibly enhance the quality of predicted items of the Bayesian classifier. Thus, we decide to apply this additional preprocessing step to improve the quality of the recommended topics.

3.1.2 CNBN-based prediction

In our previous work [8], a Multinomial Naïve Bayesian network (MNBN) has been used to extract topics, and the results were encouraging. In this paper, in an attempt to improve the recommendation capability as well as to deal better with unbalanced datasets, we adopt an enhanced version of the Multinomial Naïve Bayesian Network called Complement Naïve Bayesian Network to compute the predictions. The term "*naïve*" refers to the assumption that all the features are conditionally independent. In other

[10] https://www.eclipse.org/jgit/

[11] https://pygithub.readthedocs.io/en/latest/index.html

[12] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

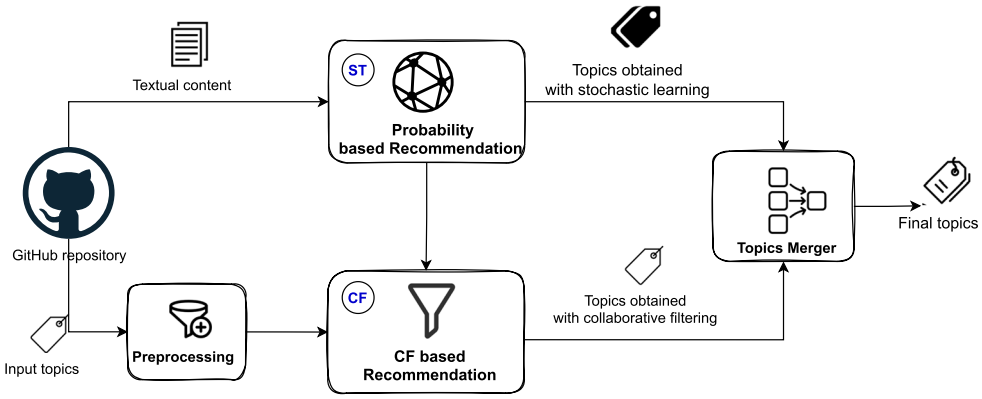

Fig. 2 Overview of the HybridRec approach

words, the network augments its prediction capabilities when each element has weak or no ties with the others. This model has been successfully employed to classify multinomial distributed data. Though MNBN and CNBN are similar from the structural point of view, their underpinning mechanisms used to compute the outcomes are completely different. In particular, MNBN employs a stochastic model that predicts a certain class **c** by relying on its training data. In contrast, CNBN makes use of the training data coming from all classes except **c**. In such a way, the performed estimation is more effective as the model uses a more even amount of training data per class. To be concrete, CNBN computes predictions by means of the following formula.[13]

$$\hat{\theta}_{ci} = \frac{\alpha_i + \sum_{j:y_j \neq c} d_{ij}}{\alpha + \sum_{j:y_j \neq c} \sum_k d_{kj}}$$

$$\begin{aligned} w_{ci} &= \log \hat{\theta}_{ci} \\ w_{ci} &= \frac{w_{ci}}{\sum_j |w_{cj}|} \end{aligned} \tag{2}$$

where the summations are over all documents j not in class **c**, d_{ij} is the tf-idf value of term i in document j and a_i is a smoothing parameter used to compute the final weight w_{ci}.

This mechanism impacts positively on the computation of the weights by using the same input data. In fact, using CNBN instead of MNBN leads to better accuracy considering unbalanced datasets [23]. After this phase, CNBN produces a list of *top-N* topics according to their probability.

[13] We made use of the Python implementation of CNBN embedded in the ***scikit-learn*** library (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html).

3.1.3 Post natural language processing

The following *lightweight post-processing* steps are performed on the obtained topics, with the aim of further enhancing the prediction capabilities:

- **Dash removal**: All the dash occurrences in each topics are removed. For instance, the *build-system* topics becomes *buildsystem*. In this way, we enlarge the possible set of recommended items;
- **Stemming**: Using Porter's stemmer implementation provided by the *nltk* language processing library,[14] we squeeze each term to its root by cutting the suffix. Thus, terms such as *monitoring* or *testing* collapse to *monitor* and *test*, respectively.

The topics retrieved by **ST** are fed as input for the collaborative-filtering component to further refine the recommendation results, as we explain in detail in the next subsection.

3.2 CF: The collaborative filtering-based component

Though CNBN can recommend relevant topics by using textual contents, it provides only *featured* topics, i.e., a set of topics curated by GitHub. Therefore, it is necessary to have machinery to provide also non-featured topics. Being inspired by the TopFilter system [7], in this work we further extend the recommendation capabilities of the **ST** component to non-featured topics, using a collaborative-filtering recommendation engine.

Figure 4 provides an overview of the **CF** component. The `Preprocessing` module is used to obtain a *filtered* dataset. Given an initial set of topics already assigned to the repository of interest, `Data Encoder` encodes it in a graph-based structure to represent the mutual relationships

[14] https://www.nltk.org/api/nltk.stem.html

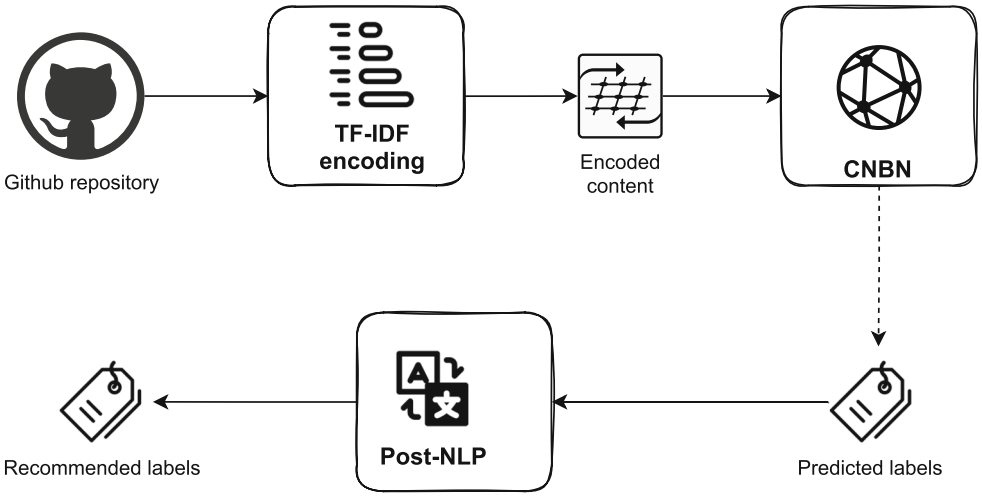

Fig. 3 The **ST** component

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 [27]. Then, `Similarity Calculator` computes similarities among all the considered artifacts. Finally, the `Recommendation Engine` module retrieves a ranked list of top-N topics that are suggested by using user-based collaborative filtering technique [35]. The functionalities and preprocessing techniques implemented in **CF** are described in detail as follows.

3.2.1 TopFilter preprocessing

We adapt semantic mapping rules proposed in a recent work [13], revise them as well as propose our own rules, with the aim of enhancing the quality of the retrieved topics. These rules have been manually defined by building direct relationships among the terms. For instance, topics *firefox* and *chrome* are rooted to a common term, i.e., *browser*. In particular, we adopt two types of rules, i.e., Aggregating Rewriting Rules (ARR)s and Extending Rewriting Rules (ERR)s. The former are used to restrict the dictionary of possible topics by rewriting the topics in a different manner, e.g., `exporter` and `exporting` are aggregated in `export` that preserves their semantic. Meanwhile, the latter provide additional information about the topic, e.g., the repository containing the `sysmodule` topic are augmented by adding the `system` and `module` topics. Given a repository, ERRs extend the list of topic by adding new ones. Specifically, we perform the following ordered preprocessing steps:

1. **[ARR - *version removal*]** All the substrings referring to versions are removed from the topics. We use the following regular expression $v[ndn.]+$ to match versions. Then, the identified string matches are removed from the topics. For instance, this rule allows us to aggregate terms like `riot-api-v4` and `riot-api-v3` to the common term `riot-api-`;
2. **[ARR - *extra digits removal*]** Punctuation digits, non-English and non-ASCII characters at the end of the topics are filtered out. This rule enables us to match topics that are very similar apart from non alphabetic chars. For instance, using this rule will boil `python3`, `python2` and `python` down to the final term `python`;
3. **[ERR - *abbreviation expansion*]** Popular software engineering and computer science related abbreviations and acronyms, e.g., lib, config, DB, doc, are rewritten with their original form, e.g., library, configuration, database, document.
4. **[ERR - *split frequent topics*]** By computing the frequency of tokens, we split those that are made from the most frequent topics. For example, `java-script-tutorial` is split into two separate terms, i.e., `javascript` and `tutorial`;
5. **[ERR - *alias substitution*]** Relying on the topic aliases recently proposed [13], we transform topics according the identified aliases. As an example, `angularjs` is converted to `angular` following this rule;
6. **[ERR - *split tokens*]** Tokens are split based on the snake_case (terms separated by underscores), camelCase (terms separated with a single capitalized letters) or kebab-case (terms separated by hyphens) naming conventions. For instance, `springDemo` is split into `spring` and `demo`;
7. **[ARR - *nlp process*]** Stop words are removed, then NLP stemming and lemmatization methods are applied on the topics. For instance, `programming-in-haskell` becomes `program` and `haskell` topics;
8. **[ARR - *infrequent topics removal*]** Finally, topics with a frequency of less than a given threshold are removed.

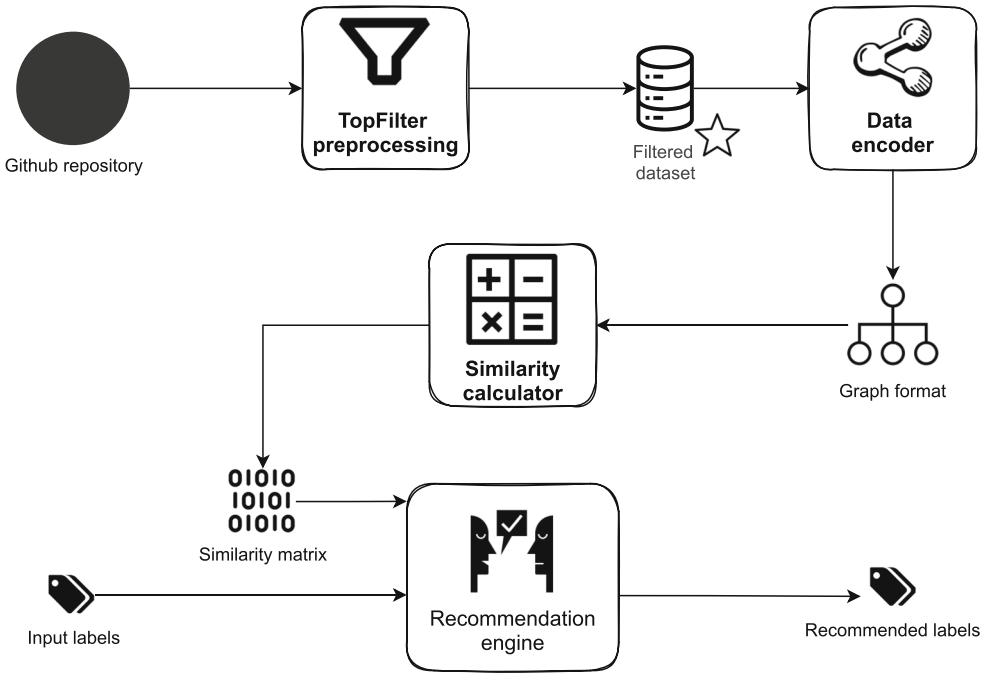

Fig. 4 The **CF** component

3.2.2 Data encoder

This component represents the relationships between projects and topics in a matrix, whose rows and columns represent all the projects and all the corresponding topics. Thus, the cell (i, j) is set to 1 if the artifact in the i^{th} row is labeled with the topic in the j^{th} column, 0 otherwise. The matrix is eventually constructed by preprocessing raw topics with the same NLP techniques used in MNBN to filter out possible biased terms (see Section 3.1.2).

To illustrate how `Data Encoder` works, we consider a set of four GitHub repositories $P = \{p_1, p_2, p_3, p_4\}$ together with a set of topics $T = \{t_1 = junit; t_2 = testing; t_3 = specs; t_4 = module; t_5 = mocking, t_6 = mock\}$. Moreover, the *repositories-topics* inclusion relationships is denoted as $\ni$. A parsing of the projects reveals 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, t_5 and t_6 collapse on the same term which is named as t_6. The final project-topic matrix is shown in Table 2.

Table 2 The *artifact-topic matrix* for the example

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

3.2.3 Similarity calculator

Given the encoded data, this module computes the similarity among the projects by considering the mutual relationships. Two nodes in a graph are considered to be similar if they share the same neighbours by considering their edges. We represent a set of projects and their labels in a graph, so as to calculate the similarities among the projects. For instance, Fig. 5 depicts the graph-based representation of the project-topic matrix in Table 2.

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

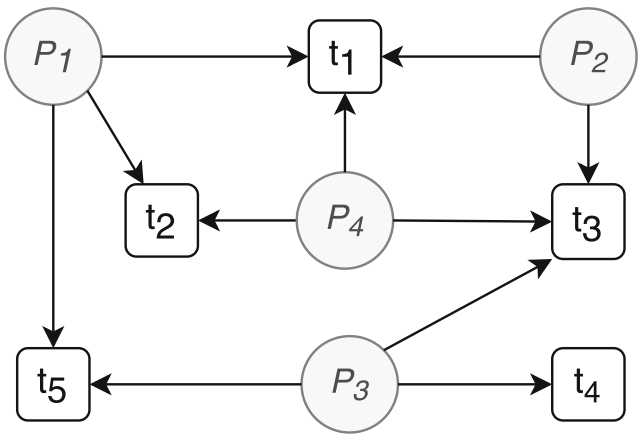

Fig. 5 Graph representation for repositories and topics

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.

$$sim(p,q) = \frac{\sum_{t=1}^{n} \phi_t \times \omega_t}{\sqrt{\sum_{t=1}^{n}(\phi_t)^2} \times \sqrt{\sum_{t=1}^{n}(\omega_t)^2}} \quad (3)$$

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

3.2.4 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. The collaborative-filtering recommender works based on the assumption that *"if projects share some topics, then they will probably share additional topics."* In other words, it predicts topics' presence by means of those collected from the *top-k* similar projects using the following formula [21]:

$$r_{p,t} = \overline{r_p} + \frac{\sum_{q \in topsim(p)}(r_{q,t} - \overline{r_q}) \cdot sim(p,q)}{\sum_{q \in topsim(p)} sim(p,q)} \quad (4)$$

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 (3).

The next sections present the experiments performed to evaluate HybridRec as well as to compare it with MNBN, TopFilter and TopFilter$^+$.

4 Evaluation

This section describes the process conducted to study the performance of HybridRec. In particular, three research questions are presented in Section 4.1 to address different aspects of the quantitative and qualitative evaluations. Afterward, an informative description of the datasets exploited in the evaluation is given in Section 4.2. Sections 4.3 and 4.4 describe the evaluation metrics and process, respectively.

To facilitate future research, we made available the HybridRec tool together with the related data in GitHub.[15]

4.1 Research questions

We study the performance of our proposed approach by answering the following research questions:

- **RQ**$_1$: *How does HybridRec perform compared to TopFilter$^+$?* In our previous work [7], TopFilter$^+$ was used to predict topics for GitHub repositories, obtaining promising results. HybridRec has been conceptualized following the same line of reasoning and implementing various boosting mechanisms. This research question investigates the impact of the enhancement on the overall prediction performance by comparing HybridRec with TopFilter$^+$;
- **RQ**$_2$: *In comparison to MNBN, does CNBN contribute to a better HybridRec performance?* We compare the recommendation capability of the two considered stochastic networks, i.e., MNBN and CNBN, using an unbalanced dataset. This aims to find out the factors that contribute to gain in the prediction performance;
- **RQ**$_3$: *How do the preprocessing steps impact on the HybridRec performance?* We investigate the impact of the preprocessing steps on the collaborative-filtering component by experimenting with both preprocessed and raw datasets;
- **RQ**$_4$: *What are the key differences between GitHub and MVN Repository, and how do they impact on the whole HybridRec recommendation process?* GitHub projects available on MVN Repository[16] are characterized by different features as well as metadata; we investigate to which extent such varieties could affect the mining process and the prediction capabilities of HybridRec.

4.2 Data extraction

To answer the four research questions, we curate four different datasets, namely D_1, D_3, D_2, and D_M as shown in Table 3 and described as follows.

- D_1 is the original dataset already used in our previous work [7, 8] consisting of 11,694 GitHub repositories with 19,337 topics;
- D_2: As discussed in our previous work [7], infrequent topics negatively affect the prediction outcomes. 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. Afterward, topics that occur in less than 20 repositories were removed, obtaining D_2 with 6,253 repositories and 455 topics;

[15] https://github.com/MDEGroup/HybridRec

[16] https://mvnrepository.com/

Table 3 Datasets features

	D_1	D_2	D_3	D_M
# of artifacts	11,694	6,253	5,620	2,932
# of topics	19,337	455	6,442	489
Avg. number of topics	8.24	6.70	8.60	3.23
Avg. frequency of topics	16.13	42.10	29.90	36.5

- D_3: To evaluate the impact of the preprocessing steps, we created the third dataset by applying the preprocessing rules presented in Section 3.2.1. The resulting D_3 dataset consists of 5,620 repositories labeled with 6,442 topics.
- D_M: To evaluate the performance of HybridRec on a different repository of artifacts, we collected a set of 2,932 unique projects from MVN Repository Repository using Beautiful Soup,[17] a well-founded Python scraping library. Tags in MVN Repository are well-maintained as they are not freely assigned by developers but by a central authority once artifacts have been made available on the platform. The D_M dataset contains the most popular tags belonging to the *top categories* curated list.

4.3 Metrics

In this work, *success rate*, *precision*, *recall*, *top rank*, and *catalog coverage* are employed to study the considered systems, as these metrics have been widely exploited by related research [21, 24]. First, the following notations are introduced:

- t is the frequency cut-off value of input labels, i.e., all topics that occur less than t times are removed from the dataset;
- τ is the number of topics fed as input to TopFilter;
- N corresponds to the cut-off value for the ranked list of recommended items;
- k specifies the number of top-similar neighbor projects TopFilter incorporated to predict 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.

[17] https://www.crummy.com/software/BeautifulSoup/

Using the aforementioned notations, the success rate, accuracy and coverage metrics are defined as follows.

Success rate@N Given a set of testing projects P, SR@N is defined as the fraction of projects having at least a matched topic among the total number of queries.

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

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 [6]:

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

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

Top rank It measures the percentage of the first top element in the user's topics:

$$Top\ rank = \frac{TpRank(r)}{|R|} \times 100\% \tag{8}$$

where $TpRank(r)$ returns 1 if the first predicted element belongs to $GT(p)$, 0 otherwise.

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.

$$coverage@N = \frac{\left|\cup_{p \in P} REC_N(p)\right|}{|T|} \tag{9}$$

4.4 Evaluation process

We use the *ten-fold cross-validation* technique [17] to analyze the performance of our proposed approach. Figure 6 depicts the evaluation process consisting of three consecutive steps, i.e., *Data Preparation*, *Recommendation Production*, and *Outcome Evaluation*, which are explained as follows.

Data preparation This phase is conducted to collect repositories from GitHub that match the requirements defined in previous section during the `Data collection` step. The dataset is then split into a training and a testing set

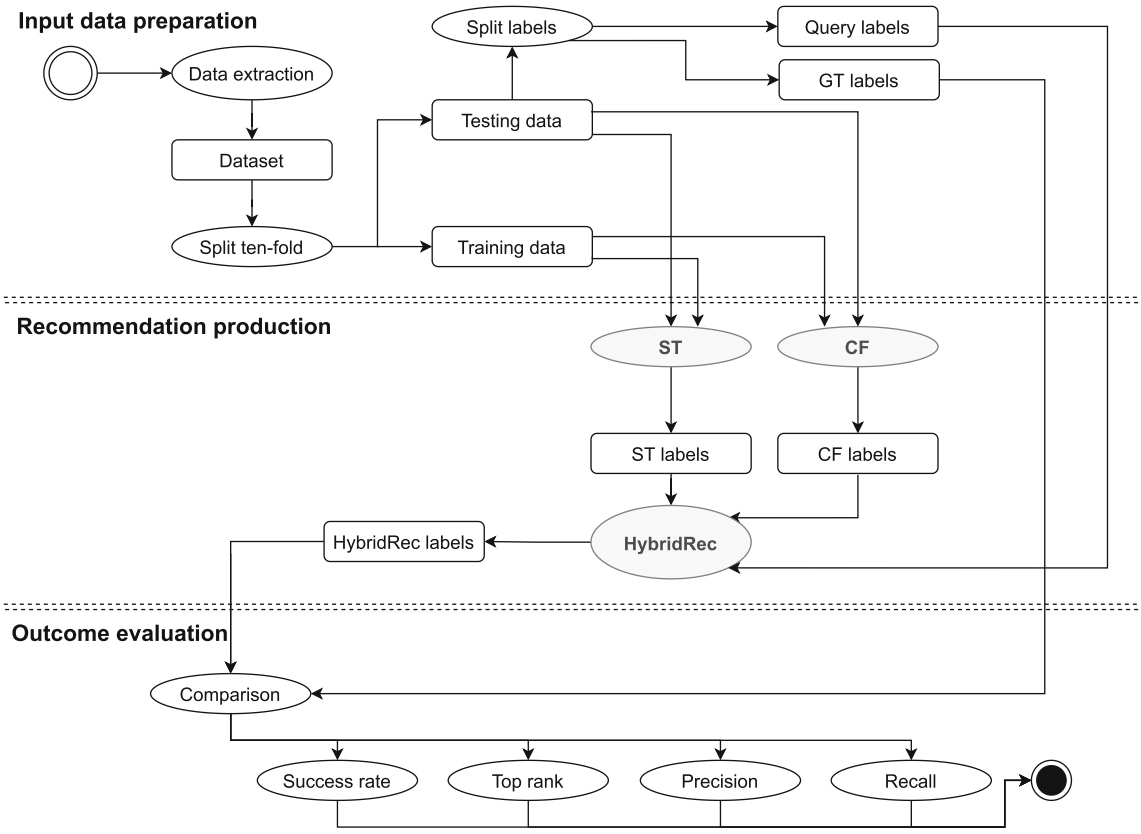

Fig. 6 Evaluation process

using `Split ten-fold`. Since the recommender systems are different in terms of input data, i.e., CNBN requires *README* files as input and training data, whilst HybridRec uses a set of assigned topics as input and for training, testing and training sets need to be specifically parsed for each individual approach. The `Split labels` activity simulates a real development scenario when a developer has already assigned some topics to her repository, and waits for recommendations, i.e., `Query labels`. For instance, given a GitHub repository and its topics, we fed the CNBN component using the repository README file annotated with the corresponding topics. Meanwhile, the testing data represent unlabeled README files that need to be categorized by the network. Contrariwise, TopFilter requires only training topics and an initial set of labels extracted from the input repository, namely `Query labels` since the underpinning engine makes use of a collaborative filtering technique that suffers from the cold start problem.

Recommendation production To enable the evaluation of HybridRec, we extracted a part of the topics for each testing project, resulting in the ground-truth data. The remaining part is used as a query to produce recommendations. We parsed and encoded text files in vectors using the TF-IDF weighting scheme to provide input to CNBN.

Outcome evaluation We evaluate HybridRec and compare it with MNBN, TopFilter, and TopFilter$^+$, analyzing the recommendation results by comparing them with those stored as ground-truth data to compute the quality metrics, i.e., *Success rate*, *Precision*, *Recall*, *Top rank*, and *Catalog coverage*.

5 Results and discussion

In this section, we study the performance of our proposed approach by answering the research questions presented Section 4.1. First, Section 5.1 takes a concrete example to illustrate how TopFilter$^+$ and HybridRec recommend topics to a repository. Afterward, Sections 5.2 and 5.3 report and analyze the experimental results.

5.1 Explanatory example

Before answering the research questions, we illustrate the recommendations provided by TopFilter$^+$ and HybridRec through a running example. As shown in Table 4, the maintainers of the repository *markdown-viewer*[18] labeled it

[18] https://github.com/simov/markdown-viewer

Table 4 Recommendation results for the *markdown-viewer* repository (topics matching with ground-truth data are reported in bold)

Original topics (D_2)	Freq. (D_2)	Processed topics (D_3)	Freq. (D_3)	Rank	TopFilter$^+$	HybridRec
javascript	865	javascript	884	1	**markdown**	**markdown**
chrome-extension	69	chrome	123	2	**firefox**	emoji
markdown	105	markdown	123	3	sass	**browser**
firefox	65	firefox	68	4	vim	**chrome**
chrome	103	extension	38	5	github-api	emacs
firefox-addon	29	viewer	34	6	c	html
firefox-extension	17	addon	38	7	editor	**javascript**
browser-extension	13	browser	220	8	**javascript**	**extension**
markdown-viewer	4			9	html	**firefox**
				10	vue	mozilla
				11	document	privacy
				12	react	safari
				13	android	golang
				14	git	opera
				15	nodejs	**addon**
				16	library	python
				17	emoji	editor
				18	web	react
				19	python	secure
				20	book	latex

with nine labels as listed on the *Original topics* column. The column *Processed topics*, shows the outcomes obtained from the preprocessing step on the original topics. The last two columns report the ranked recommendations provided by TopFilter$^+$ and HybridRec.

Moreover, for each topic, Table 4 lists the corresponding frequencies over the D_2 and D_3 datasets. For instance, the `split token` rule (see Section 3.2.1) rewrites *chrome-extension* as *chrome* and extension, while the `alias substitution` rule generates *browser* from *chrome* and *firefox*. In other words, the rewriting rules refine the mined topics to generate more exhaustive and coherent ones. For instance, it is evident that *firefox-extension*, *chrome-extension*, and *browser-extension* can be better represented by their corresponding short forms: *browser*, *firefox*, *chrome* and *extension*. By looking to the original topics and the preprocessed ones, we can see that the semantics of the topics are preserved but their frequencies are increased. TopFilter$^+$ and HybridRec return as output a ranked list of items, and we take the first top-20 topics, and match them with the ground-truth data. It is evident that HybridRec provides more matched items compared to TopFilter$^+$. In the following experiments we show that the preprocessing steps reduce the usage of different terms that have a very close semantics and increase the topic frequency.

The values of the considered quality metrics, i.e., success rate, precision, recall, top rank, and catalog coverage, reported in the next subsections are obtained by calculating their average values on all the folds from the cross-validation process.

5.2 RQ$_1$: How does HybridRec perform compared to TopFilter$^+$?

We compare HybridRec with TopFilter$^+$ on all the considered datasets, i.e., D_1, D_2, and D_3. Given a testing project p, a certain number of topics τ is used as input, and the remaining ones are saved as ground truth data *GT(p)*. In our experiments, τ is always considered half of the number of topics already assigned to the project under analysis, and the number of neighbor projects k is set to 20. This was identified as the configuration that brings the best performance to TopFilter$^+$ [7]. Moreover, we try with different values of N to find out how the size of recommended items impacts the prediction performance.

The average success rate, precision, and recall scores obtained by running the ten-fold cross-validation technique with HybridRec and TopFilter$^+$ on D_1, D_2 and D_3 are reported in Table 5, considering consecutive cut-off values, i.e., $N = \{1, \ldots, 10\}$. In the table, a better performance – corresponding to higher scores – is printed in bold. Altogether, it is evident that HybridRec outperforms TopFilter$^+$ for any value of N except the success rate for $N = 1$. For instance, the best success rate obtained by TopFilter$^+$ is 0.479 when $N = 10$, while the corresponding score

Table 5 Success rate, precision, and recall

	Success rate		Precision		Recall	
N	TopFilter$^+$	HybridRec	TopFilter$^+$	HybridRec	TopFilter$^+$	HybridRec
1	**0.171**	0.153	0.113	**0.212**	0.014	**0.034**
2	0.223	**0.236**	0.077	**0.179**	0.018	**0.056**
3	0.258	**0.291**	0.060	**0.153**	0.021	**0.070**
4	0.276	**0.317**	0.052	**0.133**	0.024	**0.082**
5	0.290	**0.335**	0.044	**0.117**	0.025	**0.090**
6	0.382	**0.504**	0.084	**0.133**	0.058	**0.122**
7	0.417	**0.549**	0.099	**0.137**	0.079	**0.145**
8	0.449	**0.579**	0.101	**0.136**	0.091	**0.164**
9	0.465	**0.598**	0.099	**0.132**	0.100	**0.179**
10	0.479	**0.614**	0.095	**0.127**	0.108	**0.191**

by HybridRec is 0.614. Moreover, the maximum precision score is 0.212 for HybridRec, which is much better than 0.112, the maximum precision achieved with TopFilter$^+$. Concerning recall, though both approaches yield a considerably low performance, HybridRec still always outperforms TopFilter$^+$ by all the cut-off values N. We find out in RQ_4 when HybridRec can improve its prediction performance.

Answer to RQ_1. In comparison to the baseline TopFilter$^+$, HybridRec yields a substantial performance improvement in terms of success rate, precision, and recall.

5.3 RQ_2: In comparison to MNBN, does CNBN contribute to a better HybridRec performance?

In our previous work [7], we utilized a balanced dataset manually curated from GitHub to improve MNBN's performance as the network is not able to handle unbalanced datasets. Nevertheless, this does not essentially resemble a real-world situation in the context of OSS platforms, where the distribution of topics and repositories is usually not balanced. Thus, we employ an enhanced version of MNBN, namely CNBN, to deal with realistic settings. This research question aims to validate such a hypothesis. To compare with MNBN, we run HybridRec using only the **ST** component (see Section 3.1), without triggering the subsequent collaborative filtering phase.

Figure 7 shows the success rate considering different cut-off values, i.e., $N = \{1, \ldots, 10\}$. It is evident that using CNBN obtains a better success rate with compared to using MNBN, given that an unbalanced dataset is considered. For instance, the improvement is around 10% with $N = 2$, i.e., MNBN and CNBN achieve 0.50 and 0.67 respectively in recommending at least one correct topic. As expected, increasing the number of recommended items leads to a better performance, i.e., CNBN's success rate reaches 0.90 with $N = 9$ and $N = 10$. This holds also for the MNBN, which however achieves worst performance since its success rate is always lower than 0.65.

To better study the performance of both techniques, i.e., MNBN and CNBN, we compute the precision and recall scores and show them in Table 6. Overall, CNBN improves the results obtained by MNBN for all the considered metrics. By considering all the cut-off values, the precision scores are increased by 10% on average. In particular, precision reaches its maximum at N=2, i.e., 26.57 and 35.00 for MNBN and CNBN, respectively; The minimum value obtained by MNBN is 9.41, while the corresponding score by CNBN is 15.44. Concerning recall, we observe a remarkable improvement when CNBN is adopted, i.e., it increases MNBN's results 30% of the time with $N = 10$. As expected, increasing the number of the returned items positively affects the performance of both networks. Still, CNBN improves recall from 40.14 to 74.07, while MNBN gains 45.38 as its maximum value. In other words, the usage of CNBN reduces the impact of false negatives during the prediction phase. Finally, the advantage of CNBN is eventually confirmed by the Top-rank metric, which measures the accuracy of the two networks in recommending the first item, i.e., the most probable one. In particular, the computation shows that CNBN obtains a better performance also in this case by increasing the top rank prediction up to 65.85 while MNBN gets only 49.55.

Answer to RQ_2. On an unbalanced dataset, compared to MNBN, CNBN improves the prediction performance. As data sources are unbalanced by their nature, adopting CNBN helps obtain a more precise prediction in the field.

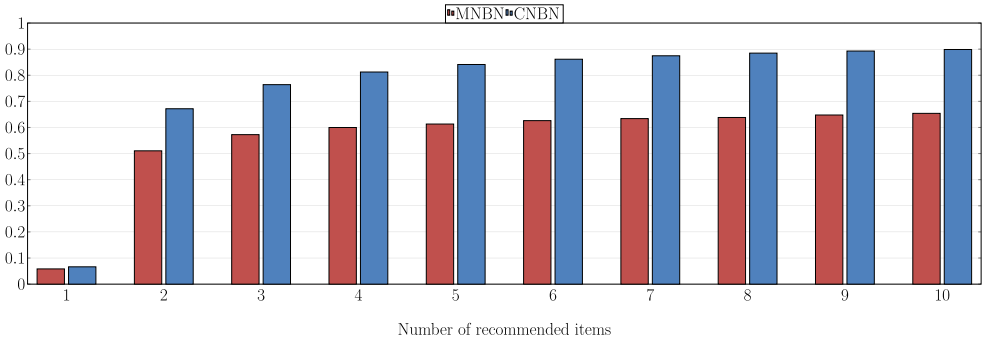

Fig. 7 Success rate for D_1 considering $N = \{1, \ldots, 10\}$

5.4 RQ$_3$: How do the preprocessing steps impact on the HybridRec performance?

We measure the impact of the preprocessing steps on the collaborative filtering recommendation engine on the three datasets collected from GitHub, i.e., D_1, D_2 and D_3. As described in Section 4.2, D_2 is obtained from D_1 by filtering out topics that occur in less than 20 repositories, while D_3 is extracted from D_1 by applying the preprocessing rules defined in Section 3.2.1. To compare with the original TopFilter approach [7], we run HybridRec using only the **CF** component (see Section 3.2).

Given a testing project p, a certain number of topics is used as input, i.e., $\tau = 5$, and the remaining ones are saved as ground truth data, i.e., *GT(p)* (cf. Section 4). Moreover, we investigate various number of recommended items $N = \{5, 10, 15, 20\}$ to understand how the size of recommended items impacts the prediction performance of TopFilter. The average success rates obtained by running the ten-fold cross-validation technique with HybridRec on D_2 and D_3 are depicted in Fig. 8.

It is evident that HybridRec yields a better success rate when preprocessed datasets, i.e., D_2 and D_3, are considered. For instance, it gets SR@1 of 0.715 for D_3, while for other datasets, the corresponding value is always lower than 0.42. Moreover, by comparing the result given by considering different cut-off values N, we realize that there is no significant difference between the results for $N = 10$ to $N = 20$. This means that most of the matched items concentrate on the top-5 ranked list, and considering a longer list of items does not bring any positive matches.

To further study the performance of HybridRec, we depict in Fig. 9 the precision/recall curves (PRCs). For this setting, we varied the recommended items N from 1 to 20, aiming to study the performance for a more detailed recommendation list. Each dot in a curve represents the precision and recall scores obtained for a specific value of N. Furthermore, we fixed $k = 20$ since this number of neighbors brings the best prediction outcomes among others, while it allows HybridRec to maintain a reasonable execution time. As a PRC close to the upper right corner corresponds to a higher precision and recall, suggesting a better performance [21], Fig. 9 shows that by considering a preprocessed dataset, HybridRec gains a better prediction. In particular, the worst precision-recall relationship is seen by the raw dataset D_1, while DGP obtains the best one. Overall, these results are consistent with those presented in Fig. 8, i.e., using the preprocessing steps given in Section 3.2.1 enables HybridRec to enhance its performance substantially.

Table 6 Precision, recall with D_1

	Precision		Recall	
N	MNBN	CNBN	MNBN	CNBN
1	0.582	**0.663**	0.323	**0.374**
2	0.266	**0.350**	0.298	**0.401**
3	0.231	**0.333**	0.362	**0.529**
4	0.194	**0.292**	0.392	**0.597**
5	0.164	**0.257**	0.408	**0.641**
6	0.143	**0.227**	0.422	**0.673**
7	0.126	**0.203**	0.431	**0.694**
8	0.112	**0.184**	0.437	**0.714**
9	0.102	**0.168**	0.447	**0.728**
10	0.941	**0.154**	0.454	**0.741**

Table 7 Catalog coverage

N	D_1	D_2	D_3
1	0.013	0.202	0.006
2	0.022	0.321	0.017
3	0.030	0.411	0.026
4	0.038	0.483	0.033
5	0.045	0.551	0.041
6	0.053	0.604	0.049
7	0.060	0.647	0.056
8	0.068	0.694	0.063
9	0.076	0.732	0.070
10	0.084	0.765	0.077

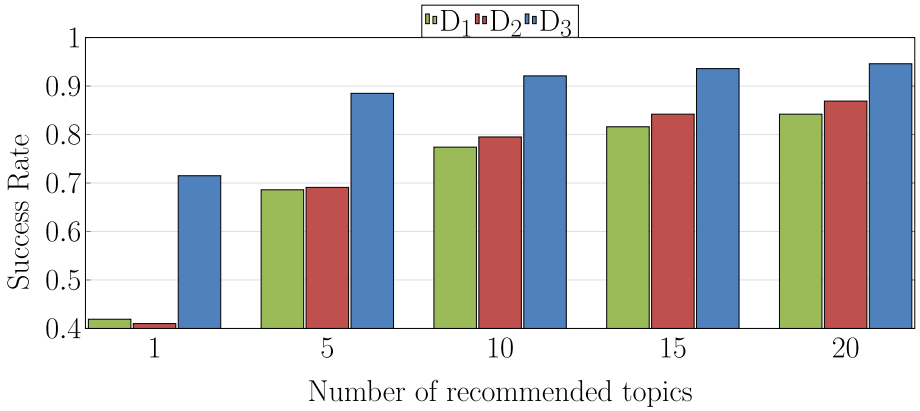

Fig. 8 Success rate values for $N = \{1, 5, 10, 15, 20\}$

Finally, we investigate if HybridRec can provide a wide range of topics to repositories, taking into consideration catalog coverage. This metric measures the percentage of the recommended topics in the training data that the model recommends to a test set, and a higher value corresponds to better coverage. Table 7 reports the average coverage values got from running HybridRec on the datasets, i.e., D_1, D_2, and D_3. The table suggests an evident outcome: we get better coverage by considering longer lists of items. We also studied the impact of the preprocessing step on the coverage results. The catalog values range from 0.013 (obtained for D_1 with $N = 1$) to 0.765 (obtained for D_2 with $N = 20$). It is important to recall that D_2 and D_3 are very different with respect to the number of topics, i.e., 455 compared to 6,442. At the same time, the sets of considered repositories are very similar (only 239 projects are discarded in D_2) (see Section 4.2). 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.

Answer to RQ$_3$. The preprocessing steps are beneficial to the recommendation process as they allow HybridRec to retrieve more relevant topics, thus improving the recommendation performance.

5.5 RQ$_4$: What are the key differences between GitHub and MVN Repository, and how do they impact on the whole HybridRec recommendation process?

To investigate HybridRec's performance in a different type of repositories, we run it on a different dataset, namely D_M collected from MVN Repository (see Section 4.2). The artifacts available in MVN Repository are labeled with *tags*, which express similar concepts to GitHub topics, i.e., summarizing their key functionalities. Moreover, each artifact comes with a textual description that can be used as a README file.

There are differences in the nature of topics in GitHub with respect to tags in MVN Repository. In particular,

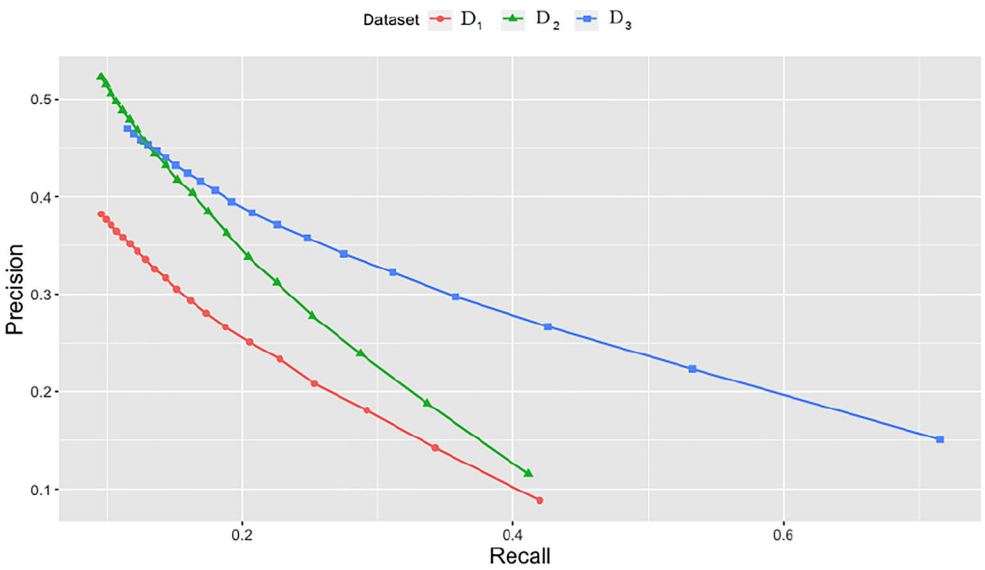

Fig. 9 Precision/recall curves

topics for a GitHub repositories are manually assigned by the owner(s). Thus, the resulting set of the given topics might include issues, e.g., duplicate drop, and word spelling, which necessarily require some pre-processing steps. In our proposed approach, we refined the mined GitHub topics by means of the rewriting rules defined in Section 3.2.1. In contrast, MVN Repository tags are well-maintained as they are not freely assigned by developers but presumably audited by a watchdog. This implies that MVN repositories are more curated than those hosted in GitHub, thus limiting errors that might happen due to the manual activities performed by developers. Altogether, this helps enhance the list of possible tags by automatically suggesting new tags, thus improving the reachability of the artifacts.

We conduct experiments for the D_M dataset using the same settings applied on the GitHub datasets and compute the quality metrics accordingly. Figure 10 represents the success rate scores obtained by running HybridRec on the Maven dataset, using different values of k, the number of neighbour projects (see Section 3.2). As we can see, the usage of a more curated dataset to make predictions contributes to improving the overall performance, i.e., the maximum success rate obtained by HybridRec reaches almost 0.90 by most of the cut-off values, and eventually, it goes beyond the 0.90 threshold with $N = 10$.

Concerning the neighborhood parameter, the results show that the best success rate is achieved with $k = 15$. This means that starting from $k = 15$, adding more neighbour projects to compute recommendations does not bring any matched tags. Overall, running HybridRec on D_M yields a better prediction performance than the results obtained with the D_1 dataset. It is worth noting that no preprocessing has been applied on Maven tags since the hand-written rules are tailored for GitHub topics that need some cleaning due to duplicates or incorrect terms. In contrast, D_M does not require such a process since tags are already fine-tuned, and their constituent terms are concretely defined. For instance, there is no need to predict the language programming tag since all projects are written in Java.

We compute precision and recall scores and show them in Fig. 11. Similar to the previous results, running HybridRec on D_M contributes to a better performance since the precision and recall are higher, compared to those obtained with the D_1 dataset in Table 5. The precision scores increase by 10% on average with the best configuration, i.e., $k = 5$. Similarly, the recall scores improve from 0.191 to 0.70 when more recommended items are considered in the ranked list.

The gain in performance is further confirmed with the catalog coverage scores in Fig. 12. HybridRec achieves a maximum catalog coverage of 0.47 with $N = 10$. This means that the probability that HybridRec recommends correct items is higher if D_M is considered. Referring to Table 3, we see that the number of topics in D_M is much smaller than that of D_1, i.e., 489 compared to 19,337. Similarly, D_M contains less projects than D_1, i.e., 2,932 compared to 11,694. Altogether, this demonstrates that even on a small Maven dataset, HybridRec can still obtain a good performance.

Answer to RQ$_4$. Compared to topics in GitHub, tags in MVN Repository are more well-defined as they are audited by a central authority. Due to this reason, running HybridRec on input data curated from Maven repositories brings in a better prediction performance.

5.6 Discussion

HybridRec has been developed by combining a complement naïve bayesian network with a collaborative-filtering technique. Moreover, we employed a tailored preprocessing phase on the considered topics to clean and refine the input data before feeding it to the recommendation engine. Altogether, the boosting mechanisms help HybridRec retrieve more relevant topics. The empirical evaluation has shown that HybridRec is able to improve the recommendation results compared to the baselines.

As discussed in Section 2.2, projects coming from GitHub exhibit different features that could affect the recommendation outcomes. In this section, by relying on the previously performed experiments, we analyze the five challenges highlighted in Section 2.2.

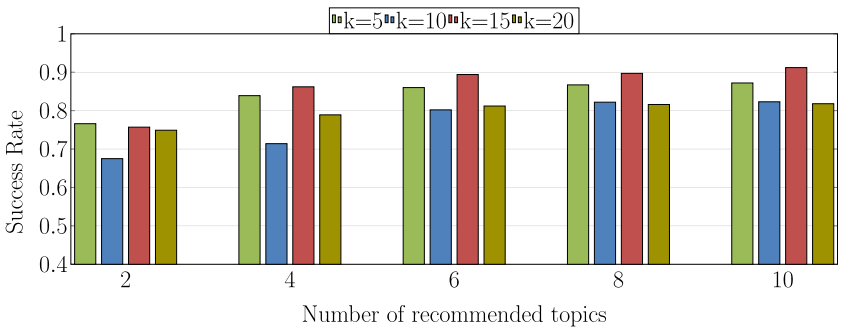

Fig. 10 Success rate on MVN Repository dataset

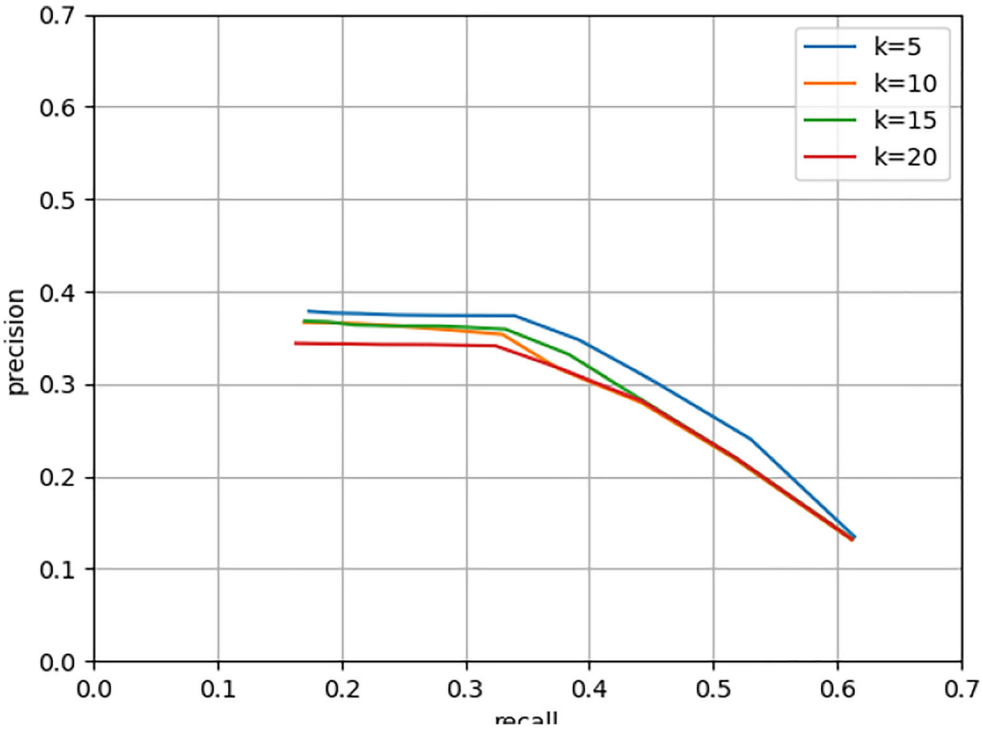

Fig. 11 Precision recall curve on the MVN Repository dataset

Concerning Challenge **C1** (*data redundancy*), we have the following investigations. Topics for a GitHub project are manually assigned by the owner(s). Thus, the resulting set of given topics might include issues (e.g., duplicate drop, and word spelling) that necessarily require some pre-processing steps. In the proposed approaches, we refined the mined GitHub topics by means of NLP techniques and word embeddings, e.g., GloVe [22], word2vec [20], and FastText [12]. Furthermore, we adopt a well-structured set of rules to consider frequent patterns as well as to refine them. We have proven that such pre-processing steps improve the overall HybridRec's prediction capabilities.

Another aspect that may impact the outcomes is the topic's *distribution*. To analyze to what extent this can affect the recommendation items, we make use of a GitHub dataset employed in our previous work [7]. Figure 13 gives an overview on the distribution of topics among repositories for the raw D_1 dataset. From this picture, we can observe that many topics are infrequent, i.e., 14,175 among 15,743 topics are used in less than 11 projects, and just a few topics are widely adopted by more than 200 projects. In other words, the long tail effect has a profound impact on the GitHub datasets.

Challenge **C2** involves the *structure of available metadata* that an OSS ecosystem can provide to perform the recommendation activities. A GitHub repository offers a lot of metadata about the activities of developers as well as their interactions. For instance, the platform keeps track of any user who makes a pull request or modifies the project by adding files through commits.[19] Furthermore, each repository is characterized by textual information, e.g., README files, or Wiki content. In other words, OSS mining tasks can be characterized by two dimensions: the *employed recommendation technique* and the proper *metadata* that can be extracted from the input data.

Table 8 describes these two dimensions in terms of the techniques and datasets presented in this work. As shown in the table, the collaborative-filtering component relies solely on the list of initial terms to perform the recommendations considering GitHub platform since all needed relations concerning projects are encoded in the matrix used to fed the recommendation engine. Furthermore, it employs the initial list of terms to enable the recommendation engine. As stated in Section 5.3, using a larger number of inputs definitively improves the tool's performance. In contrast, the **ST** component needs textual files to predict GitHub topics, namely README files.

In the context of collaborative development, the availability of *popularity mechanisms* play an important role to foster the project's visibility. As we mentioned in the description of **C3**, GitHub is reliant on a well-defined popularity mechanism that considers stars, forks, and featured topics. Both the **ST** and **CF** components benefit from using popular projects as the initial dataset, i.e., we observed the best results with respect to various quality metrics. Another

[19] https://docs.github.com/en/free-pro-team@latest/github/collaborating-with-issues-and-pull-requests/about-pull-requests

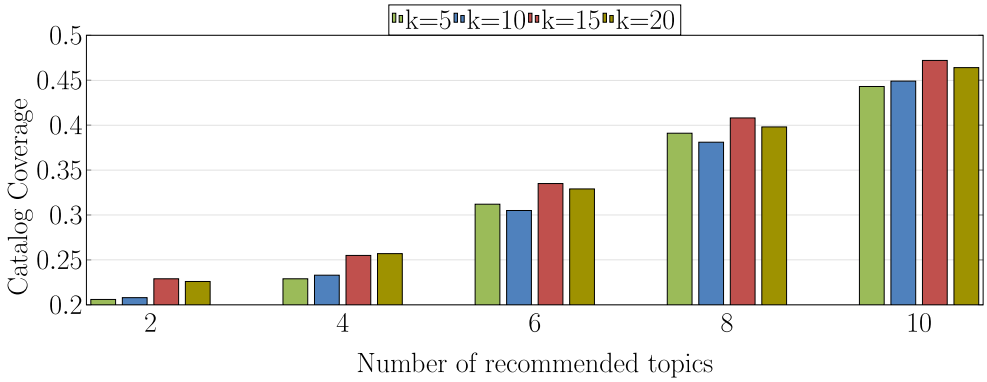

Fig. 12 Catalog coverage on the MVN Repository dataset

important factor to consider is the fact the GitHub exposes the selected featured topics in a dedicated repository[20] and web pages.[21] Despite this, the crawling phase requires a well-structured process to extract the needed information properly.

Concerning **C4** (*Crawling and Data Dump*), we see that even when the required data is available, the gathering process could be a daunting task. *Data dumps* can simplify this activity by offering all the information in a structured way, for instance with a database. As we mentioned before, data dumps for GitHub have been provided by an external tool, i.e., GHTorrent [11]. Nevertheless, data contained in these collections is not suitable for supporting automatic tagging activities. Therefore, we made use of a crawler that can download the data belonging to the repositories. In our previous studies, we exploited the PyGitHub library since GitHub offers all the required APIs.

With respect to Challenge **C5** (*Configurations of the underpinning recommendation systems*), besides the chosen OSS platform, the capabilities of the underpinning recommendation system are affected by the internal parameter configurations. The possible tuning parameters depend on the algorithm employed to perform the recommendation as well as the metadata offered by the platform. Table 9 describes the parameters employed by the two considered components, i.e., ST and CF, to bring the best prediction performance.

As described in Section 5.4, we have these values to reach a balanced dataset which is crucial to earn a better performance. This process has been conducted manually for the GitHub datasets as they exhibit heterogeneous support for each featured topic in terms of projects. Concerning the collaborative-filtering component, this consideration mainly impacts the selection of the cut-off values t. In particular, to cope with different impacts of the long-tail effect, we varied t, for example, to evaluate the **CF** component with the GitHub dataset we changed t from 5 to 20 with 5 as the step, while t has been shifted from 1 to 5 with 1 as the step. Moreover, because of the different levels of similarities between the projects of GitHub we used a different number of neighbors k when TopFilter recommends suitable labels.

6 Threats to validity

In this section, we highlight possible threats that may have an impact on the outcomes of the conducted experiments. We also list the countermeasures adopted to mitigate these issues.

Internal validity The overall recommendation capabilities of the presented approaches could be compromised by the dataset features, i.e., the number of unique tags, lack of support for each category. To cope with these risks, we analyzed several configurations obtained by means of a well-defined data preprocessing phase. These settings allow for a more comprehensive evaluation of the two approaches.

External validity As specified in Section 4.2, the data extraction phase has been conducted by relying on the GitHub API. This could compromise the quality of the obtained information as we cannot access the entire knowledge due to the rate limits. Such a threat has been mitigated by considering popular repositories. In this way, we chose the most representative projects that help minimize any potential bias.

Table 8 Metadata used by the **ST** and **CF** components

	GitHub		
Component	Topics	Featured Topics	README files
ST	✓	✓	✓
CF	✓	✗	✗

[20] https://github.com/github/explore

[21] https://github.com/topics

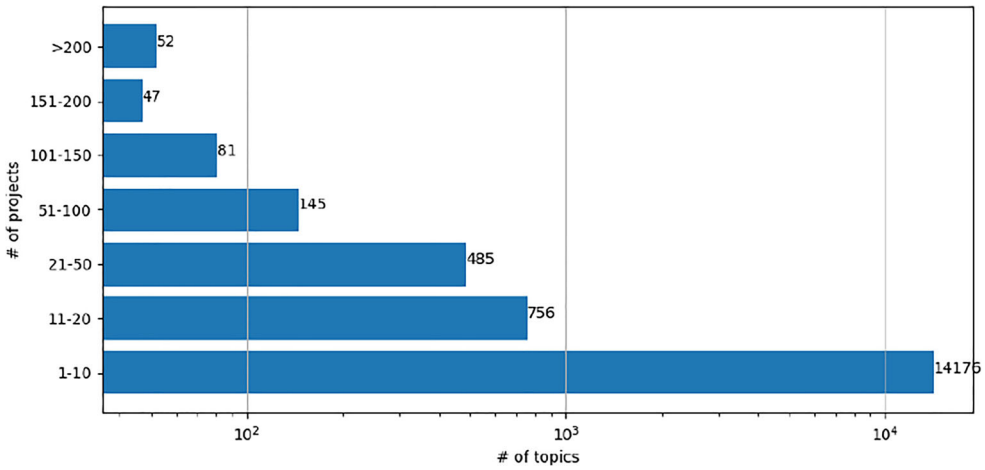

Fig. 13 The distribution of topics in the D_1 dataset

Construction validity The comparison of TopFilter$^+$ with HybridRec might be susceptible to bias. We carefully examined existing work in the domain, but no comparable tool is available with an available sound replication package. Thus, we used the same evaluation conducted to evaluate MNBN in the original work, bringing the best results. We adapted the structure of the two techniques to mitigate any possible pitfalls in the overall evaluation process.

7 Related work

In this section, we review related studies to our work, focusing on the following two main topics: *(i)* recommending GitHub topics; *(ii)* categorization of open source software projects; and *(iii)* advanced techniques for recommender systems.

7.1 Recommendation of GitHub topics

Before the introduction of topics, GRETA (Graph-Based Tag Assignment for GitHub Repositories) was the very first attempt to automatize GitHub tagging [4]. This approach exploited two concepts, namely ETG (entity-tag graph) and a random walk with a restart algorithm. The former is a graph that uses StackOverflow and GitHub data. In contrast, the latter is an algorithm that iteratively explores the global structure of the network to estimate the proximity (affinity score) between two nodes. This aims to build the graph by linking GitHub repositories with StackOverflow posts which share user and lexical similarities. Afterwards, tags are propagated using the walk algorithm.

Repo-Topix [10] was released right after the introduction of GitHub topics as a means to automatically recommend topics by relying on README files and the textual content of a given repository. Standard NLP techniques and a regression model have been applied at the early stage of the process to exclude biased terms from the recommendation process. In addition, the tool makes use of an adapted version of the Jaccard distance to enhance the quality of retrieved items further. Repo-Topix has been preliminarily evaluated using the n-gram ROUGE-1 metrics to count the number of overlapping terms between the recommendation items and the original repository description.

Recently, Izadi et al. [13] proposed Repologue, a multi-label classification tool that combines various state-of-the-art classifiers, i.e., MNB, logistic regression, FastText, and DistilBERT. The first step involves topic augmentation using hand-written association rules to derive more information from a given repository and its tags. Then, Repologue compares the mentioned models employing several word embeddings techniques to discover the best one. This can be a potential baseline for our approach. Unfortunately, the tool is not available when writing this paper.[22]

Table 9 Configuration parameters of the **ST** and **CF** components

Parameters	ST	CF
Number of considered labels	134	6,422
Number of projects	6,700	5,859
TF-IDF vectorizer	✓	–
Topic preprocessing	✓	✓
GuessLang module	✓	–
Cutoff	–	20
Neighborhood	–	25

[22]Through private communications, the authors of Repologue let us know that due to some technical issues, the entire replication package was not available to allow for a deep comparison.

GHTRec [36] exploits the BERT model to recommend personalized trending repositories[23] by using GitHub topics. First, the underpinning neural network is fed with preprocessed README files to predict topics given a repository. Afterwards, the system captures the user's topic preferences by relying on its commits. The retrieved list is eventually reranked by computing two similarity methods on the topic vectors, i.e., cosine similarity and shared similarity between the developer and a trending repository.

Sally [31] is an automated approach aiming to categorize Maven projects by relying on bytecode analysis and tags extracted from StackOverflow. Given a JAR file as the input, the tool obtains relevant data related to the project, i.e., class names, class fields, and method names. Such results are filtred considering tags excerpted from StackOverflow. In parallel, a weighted graph is computed to identify and encode project dependencies. Then, primary tags are computed by considering only the input project. In contrast, the weighted graph is used to obtain dependencies' tags. These two kinds of tags are combined to retrieve final recommendations in a tag cloud format.

Velázquez-Rodríguez and De Roover [32] have recently proposed MUTAMA, an automated multi-label tagging approach to support Maven projects. Different from our approach, the project's tags are extracted from bytecode, employing a well-founded tool. For each project, the tool takes as the input a pair of groupId-artifactId-version and the corresponding set of tags. In such a way, MUTAMA learns which projects have been tagged similarly by considering the Java classes and methods employed. Then, several machine learning algorithms provided by the MEKA tool are fed with the extracted tags and the vectors obtained from the analyzed source code.

Zhang et al. [34] solved the task of keyword-driven hierarchical classification, proposing a tool with three main modules as follows: *(i)* HIN (Heterogeneous Information Network) construction and embedding; *(ii)* key-word enrichment; and *(iii)* topic modeling and pseudo document generation. During the first step, HIN creates a graph to capture all the interactions between the core elements of GitHub repositories, like *Users*, *Words*, *Names*, to name a few. The keyword enrichment step is devised to cope with the problem of scarcity and bias of the keyword provided by users. The machinery in the last step is employed to feed a classifier with pre-labeled repositories and the overfitting related to the usages of the keywords only. The tool has been tested on two different datasets, a machine learning taxonomy with 1,600 examples and a bioinformatics one with 876 projects.

LabelGit [26] is a dataset for the classification of Java software projects. The authors proposed an approach to excerpt information directly from source code and dependency graphs. The former is obtained by a combination of techniques like *code2vec* and *fastText*, while the latter is a graph describing the dependencies between classes. Similarly, ClassifyHub [28] is a GitHub projects classification algorithm based on ensemble learning. The approach has been evaluated on a dataset consisting of 681 projects and obtained a precision of about 60% and a recall of 58%.

7.2 Automated categorization of OSS projects

With Lascad (Language-Agnostic Software Categorization and Similar Application Detection) [2], the authors proposed a tool based on information retrieval techniques, i.e., LDA (Latent Dirichlet Allocation) and hierarchical clustering. First, the tool extracts terms from the dataset's source code, followed by a refining phase where stop words and similar code-specific terms are removed. The second step is the most important one; LDA is applied on the terms corpus to get the topics, then these topics are grouped by using the hierarchical clustering technique. Finally, projects are assigned to each group and then labeled. Based on this result, during the third step, given an application as input, LASCAD retrieves corresponding software ranked by similarity.

Five different machine learning algorithms have been used to classify OSS projects [19], i.e., SVM, Naive Bayesian, Decision Tree, and IBK classifier. The training phase for each model relies on bytecode analysis performed by JClassInfo, a well-founded tool. Each model is fed with API methods, classes, and packages excerpted from 3,286 projects labeled belonging to SourceForge. The results show that SVM outperforms the other models in terms of precision, recall, and success rate. Different from our work, this study compares five machine learning models considering only 22 different categories.

Wang et al. [33] propose a tag recommendation based on a semantic graph (TRG) to classify projects extracted from two OSS communities, i.e., OhLoh and Freecode. The first step is to analyze semantic correlations between the software description and tags and encode them in a hierarchical graph. Then, a list of tags is retrieved given an input project by relying on the constructed graph. TRG eventually ranks the final list of recommendations according to their probability.

An agglomerative hierarchical clustering for taxonomy construction name AHCTC has been proposed [18] to categorize software projects stored on the OhLoh platform. To this end, the approach integrates the technique mentioned above with a topic model based on the well-established LDA algorithm to identify thematic spaces composed of similar tags. Finally, the proposed clustering technique can extract

[23] https://github.com/trending

the most central tag from the thematic space to classify the given project properly.

7.3 Boosting techniques for recommender systems

Research in recommender systems has amassed momentum in recent years, and various techniques have been proposed to equip recommender systems with the ability to retrieve highly relevant items, improving their prediction capability. In this section, we review the most notable studies in this domain, and associate them with our work.

Fan et al. [9] conceived an end-to-end framework to improve recommender systems by means of a knowledge graph, which can capture users' preferences. To this end, a user preference matrix (UPM) was built to project refined item embeddings from their latent space into the user embedding space. They evaluated the potential probability of the user preferences towards the target items through inner product results of user embedding with the projected item embedding representation. Such a technique can come in handy for the collaborative-filtering module of HybridRec, where it is used to further improve its ability to find relevant topics.

Taraghi et al. [29] presented a hybrid recommender system for open journal systems, which uses content based filtering as the basic system extended by the results of a collaborative-filtering approach. The collaborative filtering algorithm works on the basis of weighted user paths, and the authors conceived a method to reduce the undesirable effect of the navigation path, allowing the collaborative-filtering module to retrieve relevant serendipitous and novel recommendations. We anticipate that such a technique can be employed to improve the collaborative-filtering component of HybridRec, and this task is in our calendar for future research.

Tran et al. [30] provided a systematic overview of existing research on recommender systems in the healthcare domain, paying attention to different recommendation scenarios and approaches. Among the issues raised in the paper, we assume that the problem of constructing user profile is interesting and worth being investigated in the scope of the HybridRec work. We plan to improve the similarity computation by enriching projects with additional metadata such as developers, stargazers, or source code.

Given that computing similarity is important in the whole recommendation process, Al-Shamri [1] studied existing similarity modifiers and proposed various similarity modifiers that consider the active and training users' statistical parameters. In fact, computing similarity also plays a crucial role in recommender systems in software engineering [25], and we anticipate that the incorporation of new similarity algorithms – like the ones proposed by Al-Shamri [1] – into the internal design of HybridRec might help it further improve its performance.

According to a careful observation, we see that HybridRec is among the first hybrid recommender systems in software engineering, as the system works by combining different algorithms for producing recommendations. This paves the way for further improvements by considering various boosting algorithms as we have introduced in this subsection. We consider this issue as our future work.

8 Conclusion and future work

OSS platforms play an important role in aggregating and handling projects developed by a prolific community. Automatic tagging is one of the most valuable techniques to improve their discoverability, even though it requires a lot of effort to produce useful outcomes. In this paper, we conceived HybridRec, a hybrid recommender system working on top of a stochastic network and a collaborative-filtering technique to recommend topics. We performed an empirical evaluation on real-world datasets to study HybridRec by comparing it with state-of-the-art toots. The results showed that the newly conceived approach improves our former recommender systems substantially. More importantly, we demonstrated that HybridRec can increase its prediction performance on well-curated data sources.

For future work, we plan to improve the proposed framework further by analyzing the source code of OSS projects to compute similarity for HybridRec. Furthermore, we plan to conduct a well-structured user study by involving developers to evaluate HybridRec's outcomes. Last but not least, we will create a taxonomy of GitHub topics by grouping tags with a pairwise ranking algorithm. As a short-term plan, we plan to propose an integrated approach to generate a taxonomy of software domains by inspecting GitHub topics. In such an approach, GitHub repositories are fetched to collect different topics. Then, various filtering steps will be used to reduce the number of topics. Afterward, a reconciliation step is applied, where the selected topics are linked to Wikidata in order to help with the disambiguation of these terms. The final step is to create a discrete rank of the selected application domains by clustering algorithms.

Acknowledgements The research described in this paper has been partially supported by the AIDOaRT Project, which has received funding from the European Union's H2020-ECSEL-2020, Federal Ministry of Education, Science and Research, Grant Agreement n° 101007350. We thank the anonymous reviewers for their comments and suggestions that helped us improve our paper.

Funding Open access funding provided by Università degli Studi dell'Aquila within the CRUI-CARE Agreement.

References

1. Al-Shamri MYH Similarity modifiers for enhancing the recommender system performance. Applied Intelligence. https://doi.org/10.1007/s10489-021-02900-7
2. Altarawy D, Shahin H, Mohammed A, Meng N (2018) Lascad: Language-agnostic software categorization and similar application detection. J Syst Softw, 142. https://doi.org/10.1016/j.jss.2018.04.018
3. Borges H, Hora AC, Valente MT (2016) Understanding the factors that impact the popularity of GitHub repositories. In: 2016 IEEE International conference on software maintenance and evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016, pp 334–344. IEEE Computer Society. https://doi.org/10.1109/ICSME.2016.31
4. Cai X, Zhu J, Shen B, Chen Y (2016) Greta: graph-based tag assignment for github repositories. In: 2016 IEEE 40th Annual computer software and applications conference (compsac), vol 1, pp 63–72. https://doi.org/10.1109/COMPSAC.2016.124
5. Cosentino V, Luis J, Cabot J (2016) Findings from github: methods, datasets and limitations. In: Proceedings of the 13th international conference on mining software repositories, MSR '16. Association for Computing Machinery, New York, pp 137–141. https://doi.org/10.1145/2901739.2901776
6. Davis J, Goadrich M (2006) The relationship between precision-recall and ROC curves. In: Proceedings of the 23rd international conference on machine learning, ICML '06. ACM, New York, pp 233–240. https://doi.org/10.1145/1143844.1143874
7. Di Rocco J, Di Ruscio D, Di Sipio C, Nguyen P, Rubei R (2020) Topfilter: an approach to recommend relevant github topics. In: Proceedings of the 14th ACM / IEEE international symposium on empirical software engineering and measurement (ESEM), ESEM '20. Association for Computing Machinery, New York. https://doi.org/10.1145/3382494.3410690
8. Di Sipio C, Rubei R, Di Ruscio D, Nguyen PT (2020) A multinomial naïve bayesian (mnb) network to automatically recommend topics for github repositories. In: Proceedings of the evaluation and assessment in software engineering, EASE '20. Association for Computing Machinery, New York, pp 71–80. https://doi.org/10.1145/3383219.3383227
9. Fan H, Zhong Y, Zeng G, Ge C Improving recommender system via knowledge graph based exploring user preference. Applied Intelligence. https://doi.org/10.1007/s10489-021-02872-8
10. Ganesan K Topic suggestions for millions of repositories - the GitHub Blog (2017). https://github.blog/2017-07-31-topics/
11. Gousios G, Spinellis D (2012) Ghtorrent: Github's data from a firehose. In: 2012 9th IEEE Working conference on mining software repositories (MSR), pp 12–21. IEEE
12. Grave E, Bojanowski P, Gupta P, Joulin A, Mikolov T (2018) Learning word vectors for 157 languages. In: Proceedings of the eleventh international conference on language resources and evaluation (LREC 2018). European Language Resources Association (ELRA), Miyazaki. https://www.aclweb.org/anthology/L18-1550
13. Izadi M, Heydarnoori A, Gousios G (2021) Topic recommendation for software repositories using multi-label classification algorithms. Empir Softw Eng 26(5):93. https://doi.org/10.1007/s10664-021-09976-2
14. Jiang J, Lo D, He J, Xia X, Kochhar PS, Zhang L (2017) Why and how developers fork what from whom in GitHub? Empir Softw Eng 22(1):547–578. https://doi.org/10.1007/s10664-016-9436-6
15. Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, Damian D (2014) The promises and perils of mining GitHub. In: Proceedings of the 11th working conference on mining software repositories - MSR 2014. ACM Press, Hyderabad, India, pp 92–101. https://doi.org/10.1145/2597073.2597074
16. Kibriya AM, Frank E, Pfahringer B, Holmes G (2005) Multinomial naive bayes for text categorization revisited. In: Webb GI, Yu X (eds) AI 2004: advances in artificial intelligence. Springer, Berlin, pp 488–499
17. Kohavi R et al (1995) A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Ijcai, vol 14, Montreal, pp 1137–1145
18. Li X, Wang H, Yin G, Wang T, Yang C, Yu Y, Tang D (2012) Inducing taxonomy from tags: an agglomerative hierarchical clustering framework. In: Zhou S, Zhang S, Karypis G (eds) Advanced data mining and applications. Springer, Berlin, pp 64–77
19. Linares-Vásquez M, Mcmillan C, Poshyvanyk D, Grechanik M (2014) On using machine learning to automatically classify software applications into domain categories. Empir Softw Engg 19(3):582–618. https://doi.org/10.1007/s10664-012-9230-z
20. Mikolov T, Sutskever I, Chen K, Corrado G, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Proceedings of the 26th international conference on neural information processing systems - volume 2, NIPS'13. Curran Associates Inc., Red Hook, pp 3111–3119
21. Nguyen PT, Di Rocco J, Di Ruscio D, Di Penta M (2020) CrossRec: supporting software developers by recommending third-party libraries. J Syst Softw 161:110,460. https://doi.org/10.1016/j.jss.2019.110460, http://www.sciencedirect.com/science/article/pii/S0164121219302341
22. Pennington J, Socher R, Manning C (2014) GloVe: global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). Association for Computational Linguistics, Doha, pp 1532–1543. https://doi.org/10.3115/v1/D14-1162, https://www.aclweb.org/anthology/D14-1162
23. Rennie JDM, Shih L, Teevan J, Karger DR (2003) Tackling the poor assumptions of naive bayes text classifiers. In: Proceedings of the twentieth international conference on international conference on machine learning, ICML'03, pp 616–623. AAAI Press
24. Robillard M, Walker R, Zimmermann T (2010) Recommendation systems for software engineering. IEEE Softw 27(4):80–86. https://doi.org/10.1109/MS.2009.161
25. Di Rocco J, Di Ruscio D, Di Sipio C, Nguyen PT, Rubei R (2021) Development of recommendation systems for software engineering: the CROSSMINER experience. Empir Softw Eng 26(4):69
26. Sas C, Capiluppi A. (2021) Labelgit: a dataset for software repositories classification using attributed dependency graphs
27. Schafer JB, Frankowski D, Herlocker J, Sen S (2007) The adaptive web. chap. Collaborative filtering recommender systems. Springer, Berlin, pp 291–324. http://dl.acm.org/citation.cfm?id=1768197.1768208
28. Soll M, Vosgerau M (2017) Classifyhub: an algorithm to classify github repositories, pp 373–379. https://doi.org/10.1007/978-3-319-67190-1_34

29. Taraghi B, Grossegger M, Ebner M, Holzinger A (2013) Web analytics of user path tracing and a novel algorithm for generating recommendations in open journal systems 37(5):672–691. https://doi.org/10.1108/OIR-09-2012-0152, Publisher: Emerald Group Publishing Limited
30. Tran TNT, Felfernig A, Trattner C, Holzinger A (2020) Recommender systems in the healthcare domain: state-of-the-art and research issues 57(1):171–201. https://doi.org/10.1007/s10844-020-00633-6
31. Vargas-Baldrich S, Linares-Vásquez M, Poshyvanyk D (2015) Automated tagging of software projects using bytecode and dependencies. In: 2015 30th IEEE/ACM international conference on automated software engineering (ASE), pp 289–294. https://doi.org/10.1109/ASE.2015.38
32. Velázquez-Rodríguez C, Roover CD (2020) 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), pp 254–258. https://doi.org/10.1109/SCAM51674.2020.00034, ISSN: 2470-6892
33. Wang T, Wang H, Yin G, Ling CX, Li X, Zou P (2014) Tag recommendation for open source software. Front Comput Sci 8(1):69–82. https://doi.org/10.1007/s11704-013-2394-x
34. Zhang Y, Xu F, Li S, Meng Y, Wang X, Li Q, Han J (2019) Higitclass: keyword-driven hierarchical classification of github repositories
35. Zhao ZD, Shang Ms (2010) User-based collaborative-filtering recommendation algorithms on hadoop. In: Proceedings of the 2010 third international conference on knowledge discovery and data mining, WKDD '10. IEEE Computer Society, Washington, DC, pp 478–481. https://doi.org/10.1109/WKDD.2010.54
36. Zhou Y, Wu J, Sun Y (2021) Ghtrec: a personalized service to recommend github trending repositories for developers. In: 2021 IEEE International conference on web services (ICWS), pp 314–323. https://doi.org/10.1109/ICWS53863.2021.00049

Juri Di Rocco is an assistant professor at the University of L'Aquila, Italy. He obtained a PhD in Computer Science from the University of L'Aquila. He is interested in several aspects of software language engineering and Model Driven Engineering (MDE) including domain specific modelling languages, model transformation, model differencing, modelling repositories and mining techniques. More information is available at http://www.di.univaq.it/juri.dirocco.

Davide Di Ruscio is an Associate Professor at the DISIM - University of L'Aquila. His main research interests are related to several aspects of Software Engineering, Open Source Software, and Model Driven Engineering (MDE) including domain specific modelling languages, model transformation, model differencing, coupled evolution, and recommendation systems. He has published more than 170 papers in various journals, conferences and workshops on such topics. He is a member of the steering committee of the International Conference on Model Transformation (ICMT), of the Software Language Engineering (SLE) conference, of the Seminar Series on Advanced Techniques & Tools for Software Evolution (SATTOSE), of the Workshop on Modelling in Software Engineering at ICSE (MiSE) and of the International Workshop on Robotics Software Engineering (RoSE). He is in the editorial board of the International Journal on Software and Systems Modeling (SoSyM), of IEEE Software, of the Journal of Object Technology, and of the IET Software journal. More information is available at http://people.disim.univaq.it/diruscio/.

Claudio Di Sipio is a PhD student at the University of L'Aquila, Italy. He is working on mining techniques to analyse open source software and he is also investigating the application of low-code platforms to support the development of recommendation systems.

Phuong T. Nguyen obtained a PhD in Computer Science from the University of Jena, Germany. Since graduation, he has worked as a university teaching and research assistant in Vietnam and Italy. He is now with the University of L'Aquila, Italy as a tenure track assistant professor. His research interests include Computer Networks, Semantic Web, Recommender Systems, and Machine Learning. Recently, he has been working to develop recommender systems in Software Engineering for mining open source code epositories.

Riccardo Rubei is a PhD student at the University of L'Aquila, Italy. He is working on mining techniques to analyse open source software with the aim of providing developers with useful real-time recommendations.