A Multinomial Naïve Bayesian (MNB) Network to Automatically Recommend Topics for GitHub Repositories

Claudio Di Sipio, Riccardo Rubei, Davide Di Ruscio, Phuong T. Nguyen
Università degli studi dell'Aquila, Via Vetoio 2, 67100 – L'Aquila, Italy
{claudio.disipio,riccardo.rubei,davide.diruscio,phuong.nguyen}@univaq.it

Abstract

GitHub has become a precious service for storing and managing software source code. Over the last year, 10M new developers have joined the GitHub community, contributing to more than 44M repositories. In order to help developers increase the reachability of their repositories, in 2017 GitHub introduced the possibility to classify them by means of topics. However, assigning wrong topics to a given repository can compromise the possibility of helping other developers approach it, and thus preventing them from contributing to its development.

In this paper we investigate the application of Multinomial Naïve Bayesian (MNB) networks to automatically classify GitHub repositories. By analyzing the README file(s) of the repository to be classified and the source code implementing it, the conceived approach is able to recommend GitHub topics. To the best of our knowledge, this is the first supervised approach addressing the considered problem. Consequently, since there exists no suitable baseline for the comparison, we validated the approach by considering different metrics, aiming to study various quality aspects.

1 Introduction

The open source software (OSS) community makes a daily usage of open source repositories to contribute their work as well as to access to projects coming from other developers. GitHub is one of the most well-known platforms that aggregate these projects and render possible the exchange of knowledge among the users. In order to aid information discovery and help developers identify projects that can be of their interest, GitHub introduced *topics*. They are words used to characterize projects, which thus can be annotated by means of lists of words that summarize projects' features. Thanks to the availability of *topics*, several applications are enabled, including the automated cataloguing of GitHub repositories [22], further than allowing developers to explore projects by type, technology, and more.

To our best knowledge, assigning the right topics to GitHub repositories is a crucial step that, if not properly done, may hamper their discoverability. In 2017, GitHub presented *repo-topix*, a topic suggestion tool essentially based on information retrieval techniques [9]. Although the mechanism works well so far and it has been fully integrated into GitHub, in our opinion there is still some room for improvement, e.g., in terms of the variety of the suggested topics, novel data analysis techniques, and the investigation of new recommendation strategies.

In this work, we present our approach to automatically recommend topics for GitHub repositories. By exploiting a well-founded Machine Learning technique, we investigate the usage of a probabilistic model-based tool to recommend topics for a specific project by means of its README file(s) and source code. Informally, the question the proposed system can answer is:

"Which tags should I use to annotate this new project being managed by means of GitHub?"

The proposed approach employs a Multinomial Naïve Bayesian (MNB) network to extract README files' content, source code and eventually to recommend topics. It is fed with a TF-IDF vectorization of README files, which represent the most frequent terms over all documents. As the final output, the tool retrieves the best-ranked topics according to their probabilities, and suggests them to developers.

To the best of our knowledge, there are no comparable approaches that deal with the same problem, apart from the GitHub internal framework.[1] Thus, we assess the quality of the results by means of suitable metrics for the examined domain. The main contributions of the paper are summarized as follows:

- Design and implementation of a tool for the automatic recommendation of topics for GitHub repositories;
- An empirical evaluation of the proposed approach exploiting a dataset collected from GitHub;
- We made available online the tool and the dataset used in our evaluation to facilitate future research [23].

This paper is organized as follows: Section 2 provides an overview of GitHub topics and the main open challenges in the given domain. Section 3 presents our proposed approach to recommend topics for GitHub repositories. In Section 4,

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.

[1]Unfortunately, the source code of *repo-topix* is not available and, thus, it cannot be used directly as baseline.

we describe the evaluation process, and we present the experimental results in Section 5 afterwards. Section 6 discusses possible threats to validity of the findings. We present related work and conclude the paper in Section 7 and Section 8, respectively.

2 Motivations and Background

When using OSS repositories, users can be interested in acquiring knowledge from existing developed software projects [17]. However, especially in the case of large source code repositories, the potential benefits related to the availability of reusable projects might be missed if they cannot be suitably discovered. To mitigate such problems, in 2017 GitHub introduced the possibility of assigning tags[2] to projects with the final aim of increasing their discoverability. By means of such a feature, users can find and contribute to software projects by searching the topics of interest, affinity, and other relevant elements.

Figure 1 shows an example of repository stored in GitHub. It is a related to the *bootstrap* project[3], and according to the given tags, it is a `css-framework`, which involves `javascript`, `html`, and `css` artifacts among others. GitHub maintains a curated list of projects, which are organized with respect to the list of *featured topics*[4], which are meant to be the most popular and active topics. Thus, users can monitor the community's trend by consulting such a public list.

Manually assigning topics can be an error-prone activity that can lead to wrongly specified tags. Over the last years, several attempts have been made to *classify* GitHub projects by automatically inferring appropriate topics. In the context of data mining, *classification* is one of the critical operations that are used to dig deep into available data for gaining knowledge and for identifying repetitive patterns [15].

Sharma *et al.* [22] present an approach based on *topic modeling* techniques to create categories of GitHub projects. Manual interventions are needed to refine initial sets of categories, which are identified by an LDA-GA technique, that combines two algorithms: Latent Dirichlet Allocation (LDA) and Genetic Algorithm (GA) [19]. The approach proposed [22] is unsupervised, meaning that the categories of the catalogue being identified are not known ex-ante.

In a GitHub blog post [9], the author presents *repo-topix*, a tool to recommend topics for GitHub repositories. Such a tool combines standard NLP techniques to find an initial set of topics, by parsing the README files and the textual content of a repository e.g., the repository's description. Then, the results are weighted with the TF-IDF scheme and "bad" topics are removed exploiting a regression model. Using this refined list, repo-topix computes a custom version of the Jaccard distance to identify additional similar topics. To assess the quality of the framework, a rough evaluation was conducted based on ROUGE-1 metrics, an n-gram overlap metric that counts the number of overlapping units between the suggested topics and the repository description. Nevertheless, neither the tool nor the dataset is made available, thus a comparison with the approach is not feasible.

With the aim of providing a practical solution to the problem of recommending GitHub topics, in the next section we propose a novel supervised machine learning approach based on a multinomial naïve bayesian network. The challenges that we had to cope with for evaluating its performance are mainly the following ones:

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

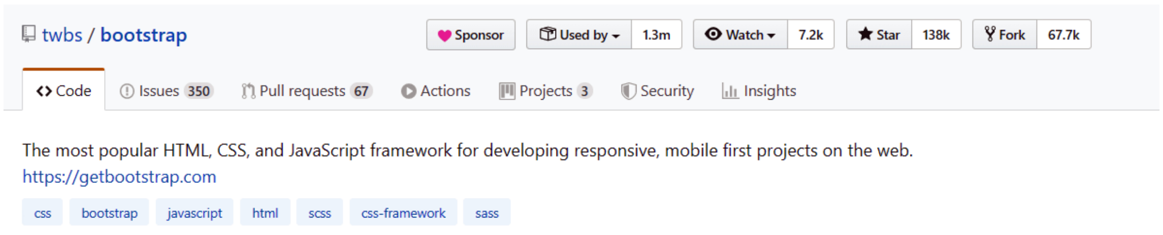
▹ *Topics distribution:* although tags can be assigned only by the owners of GitHub repositories, users may potentially wrongly specify topics or introduce information overload by inserting too many elements. Thus, creating a reliable ground truth to assess the classification performance of the proposed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

3 Proposed Approach

Given a GitHub repository, our approach aims to suggest a set of relevant featured topics by analyzing its README file and its source code. A typical scenario involves a developer who looks for suggestions while she is working with the repository. Our approach helps increase the visibility of the repository on the platform by suggesting pertinent topics. Different from other GitHub project classifiers, we turn from a standard classification to a multi-classification problem by using different modules of the source code classifier (SCC) infrastructure of the *scikit-learn* library.[6] In particular, the

[2]For the sake of presentation, the terms "tags" and "topics" are used interchangeably throughout the paper
[3]https://getbootstrap.com/
[4]https://github.com/topics

[5]http://ghtorrent.org/
[6]http://scikit-learn.org



The most popular HTML, CSS, and JavaScript framework for developing responsive, mobile first projects on the web.
https://getbootstrap.com

css bootstrap javascript html scss css-framework sass

Figure 1. An example of GitHub repository and corresponding topics.

MultinomialNB (MNB)[7] and TF-IDF vectorizer[8] components are used as follows: *(i)* we first create a curated dataset by selecting README files for each selected featured topic; *(ii)* then, the MNB is trained with vectors computed by the TF-IDF vectorizer; *(iii)* once we have the predicted list of topics, we discover the programming language with the GuessLang tool[9]; *(iv)* finally, we combine the results and deliver the list of recommended topics.

The architecture of the proposed approach is shown in Figure 2 and described in greater detail in the following subsections.

3.1 GitHub Crawling

To create a dataset with README files coming from GitHub projects, we used PyGithub [1], a library that provides a complete set of APIs to interact with GitHub repositories. Using a project's name as query, we can retrieve different types of information such as commits, issues, and name of the repository's owner. For each repository, we downloaded only its README file(s) to provide input to the MNB network. The rationale behind the selection is that README files provide an informative description of the project being developed[10], e.g., among others, what the project does, or how developers can start with it [20]. In this sense, we assume that information coming from README files is suitable for the recommendation process.

Concerning the topics, we consider only the featured ones for two reasons: *(i)* it is not feasible to train a network for all possible topics available on the platform; and *(ii)* the featured topics are the most popular according to GitHub's statistics; this means that they can offer a pretty good coverage of the community's interests. Among these topics, we deployed an additional filter on the query to limit the number of topics considered in the training as this phase suffers from a decrease in performance with too many categories. By using the GitHub query language [2], we applied the following query filter:

$$Qf = \text{"}is : featured\ topic : t\ stars : 100..80000\ topics :>= 2\text{"} \quad (1)$$

to consider only GitHub repositories having a number of stars between 100 and 80,000, and tagged with at least two topics. We set such a filter after several query refinements, as the chosen featured topics have different degrees of popularity. For instance, the most starred project of the *3d* topic has around 53,000 stars; reversely, some other topics have top rank projects that do not reach 1,000 stars. Thus, we tried to select the best query filter to include a sufficient number of repository for each topic. By imposing such a filter, we tried to avoid skewed repositories that may not have an informative README, or projects that are already abandoned for a long time.

GitHub developers use *stars* as a voting mechanism to foster the popularity of a certain project [6]. Through this system, each user can support her/his favorite projects available on the platform. *Forking* is another index related to the quality of the project. This feature is typically employed as a starting point for a new project [11]. Furthermore, there is a strong correlation between forks and stars [5]. In this sense, we suppose that a project with a high number of stars means that it gets attention from the OSS community, and thus being suitable to identify popular repositories [4, 7].

3.2 README Encoding

Before the training process, we encode the content of each README using the TF-IDF vectorizer provided by scikit-learn. The library embeds all the Natural Processing Language (NPL) techniques to preprocess README files, i.e., stop word removal, stemming and lemmatization. After the preprocessing phase, the vectorizer computes the TF-IDF weighting scheme to find the most representative terms over all documents. The TF-IDF computes the inverse document-frequency using the formula showed in Equation 2:

$$\mathrm{idf}(t) = \log \frac{1+n}{1+\mathrm{df}(t)} + 1 \quad (2)$$

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. The encoding is applied to the entire content of each README file to be fed to the MNB network.

[7]https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
[8]https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[9]https://pypi.org/project/guesslang/
[10]https://help.github.com/en/github/creating-cloning-and-archiving-repositories/about-readmes

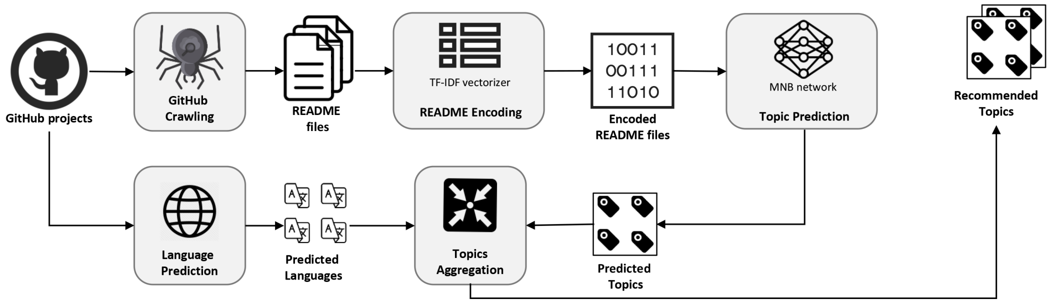


Figure 2. Overview of the proposed approach.

This is a preparatory phase to the training and it is conducted only at the beginning of the process.

3.3 Topic Prediction

Once we encoded the data using the mentioned techniques, we feed the model to perform the training phase. Naïve Bayesian network is a probabilistic model based on the Bayesian theorem that expresses the probability of a certain event given a set of preconditions [12]. The terms "Naïve" refers to the assumption that all the features are conditionally independent. This means that the classifier reaches a higher performance if each class does not have any relationship with the others. However, this condition does not always hold in practice and the model is used with relevant results. In our work, we use a Naïve Bayesian network based on a multinomial distribution, defined as follows:

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y) \tag{3}$$

where $x_1, \ldots, x_n$ are the features and y the class to be predicted with a certain probability P. We decided to employ the technique as it has demonstrated itself to outperform other ML models in the context of text classification [3]. To improve the performance of the network, we applied the TF-IDF scheme on the input data (see Section 3.2). Such a preprocessing phase should possibly enhance the quality of predicted items of the Bayesian classifier as this has been confirmed by an existing work [13].

As previously mentioned, we exploited the MNB implementation provided by scikit-learn. By default, MNB predicts only one class for each sample, we modified this feature by ranking all the results to include the most probable topics as we aim to recommend more than one topic.

3.4 Language Prediction

Among featured topics, 20 of them are related to the main programming languages, e.g., Java, Python, C. Detecting them in a GitHub project requires a comprehensive analysis of the source code. Consequently, the MNB network is not suitable for this type of analysis due to its internal construction. However, since language is an important tag, we attempted to recommend it by employing GuessLang, a Python library which was tailored for this purpose. As it has been claimed in the documentation, GuessLang can predict 20 different programming languages with satisfying accuracy. This component is executed as stand-alone instance to predict the language for each analyzed repository. To correctly predict the programming language, this module analyzes all files within a repository. Due to the timing and memory constraints in the computation, we limit the number of analyzed files to 1,000 for each repository that does not exceed 10KB in size.

3.5 Topics Aggregation

The final step consists of combining the predicted topics coming from the MNB network and the language discovered by the tool. To aim for reliable results, we replace the last element of the predicted topic list with the programming language delivered by GuessLang. Let $T = t_1, t_2, \ldots.., t_n$ be the list discovered by the MNB model, and t_l be the topic related to a programming language, then as the last element of T is the less probable of the retrieved set, the final output is the list $T_f = t_1, t_2, \ldots., t_l$, where t_l becomes the new last element.

4 Evaluation

This section presents the evaluation conducted to study our proposed approach. In particular, Section 4.1 presents the research questions we wanted to answer by means of the performed experiments and thus, to study the approach's performance. Section 4.2 introduces the datasets we populated to serve as input for the experiments. The metrics used to study the final outcomes are presented in Section 4.3.

4.1 Research questions

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

- **RQ**$_1$: *How does the variation of training data impact on the prediction performance?* To answer this question, we varied the dimension of the dataset to include more training data. In particular, we assess the quality of three different datasets to find out the best one in terms of success rate.

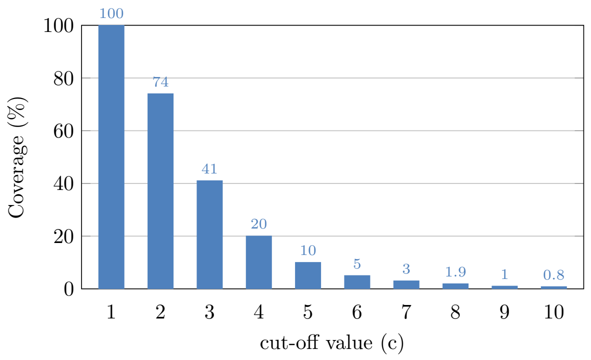


Figure 3. Distribution of the number of topics in the dataset.

- **RQ$_2$**: *Is the approach able to provide consistent recommendations considering featured topics?* We compute the metrics given in Section 4.3 to measure the relevance of our suggested topics considering the distribution of the considered repositories.

4.2 Dataset

To build the dataset, the quality filter discussed in Section 3.1 was exploited to select 134 different featured topics. The final aim is to build a balanced dataset in which every topic does not surpass the others in terms of influence, so as to avoid possible negative impacts on the final results.

The employed MNB network is slightly different from other models because it works on probability. Thus, we want to measure how the dimension of training data can affect the accuracy of the model. In this way, we created a global dataset of 13,400 README files by considering 100 repositories for each topic. Moreover, since we can recommend 20 additional topics related to the programming languages exploiting the GuessLang module, summing up our approach is able to predict a total number of 154 topics.

The number of topics for each repository has an important impact on the recommendation quality. As several repositories may have a low number of featured topics, it is necessary to examine the distribution of number of featured topics over the repositories. If we call c as the cut-off value for the number of topics, then Figure 3 displays such the distribution by varying c. The y-axis represents the percentage of repositories with respect to the created golden dataset, while the x-axis corresponds to c. We can see that, if we set the cut-off value $c = 1$, then 100% of the repositories have at least 1 featured topic. However, the percentage of repositories decreases considerably when c increases. For instance, only 40% of the repositories have c=3 featured topics and only 10% of them have c=5 featured topics. Given the circumstances, the prediction of more than 5 featured topics becomes a hard task considering this strict assumption.

4.3 Metrics

To assess the proposed approach, we exploited *success rate*, *precision*, *recall*, and *top-rank* as they have been widely used to evaluate recommender systems in Software Engineering [21]. Before going in detail, we consider the following definitions:

- *True positive (Tp)*: the topics that are correctly recommended;
- *False positive (Fp)*: the recommended topics which actually do not belong to the ground-truth data;
- *False negative (Fn)*: the topics that should be present in the recommended items, but they are not.

As the ground truth data *UsrTp(r)*, we consider all the featured topics of the real repository r available on GitHub. Then, the metrics are defined as follows:

Success rate: Given a set of R testing repositories, this metric measures the rate at which a recommendation engine can return at least 1 correct tag for each repository r.

$$Success\ rate = \frac{count_{r \in R}(Tp > 0)}{|R|} \times 100\% \tag{4}$$

where the function *count()* is a boolean function that returns 1 for each true positive.

Precision: the metric measures the rate of correct items over the entire set of recommended items:

$$Precision = \frac{Tp}{Tp + Fp} \tag{5}$$

Recall: the ratio of the user's topics appearing in the N recommended items:

$$Recall = \frac{Tp}{Tp + Fn} \tag{6}$$

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

$$Top\ rank = \frac{TpRank(r)}{|R|} \times 100\% \tag{7}$$

where *TpRank(r)* returns 1 if the first predicted element belongs to *UsrTp(r)*, 0 otherwise.

4.4 Experimental settings

We performed ten-fold cross-fold validation [14] on all the datasets. In particular, the README files used in each dataset are divided into 10 equal parts. To preserve a balanced training set along with all the training phases, for each topic, we used 90% and 10% of the files for training and testing, respectively.

Figure 4 depicts the testing process for a single test file. In particular, we computed the TF-IDF vectors for each test README file to get predicted topics. Then, we retrieved the real topics from GitHub projects. Using the API provided by the PyGithub library, we mapped each repository to its real topics. From these, we filtered out the ones that are not featured topics because the network is not able to retrieve hand-made topics out of the train set. To aim for a better coverage, some lightweight NLP techniques have been applied to both the predicted topics and user topics, i.e., stemming and dashes removal. This processing phase has no impacts

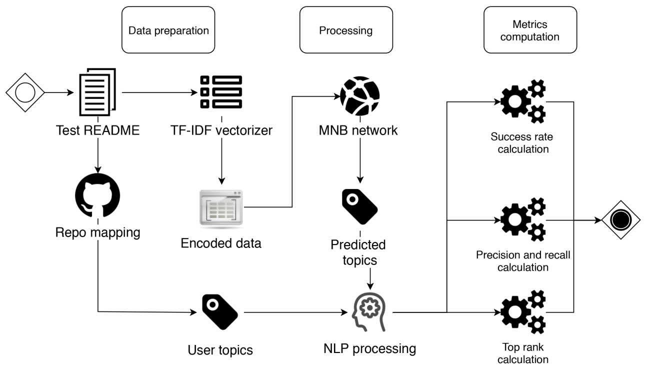


Figure 4. The testing phase for a single project.

on the topics semantic, as the employed techniques work on syntactical aspects. For example, the topics *data-structures* and *datastructures* are semantically the same and we consider them as a true positive during the accuracy measurement. The last step involves the comparison of real topics and the recommended ones. To have a qualitative measure of the approach, we computed all the evaluation metrics for each test file and we are going to report the results in the next section.

5 Results

In this section, we present the results obtained from the performed experiments. Section 5.1 describes a concrete example of 10 repositories and their corresponding topics, as well as the topics recommended by our proposed approach. In Section 5.2, we analyze the results by answering **RQ**$_1$ and **RQ**$_2$, aiming to study the approach's performance.

5.1 Explanatory example

Before going into detail, we take a concrete example to illustrate how the system recommends topics in Table 1. In this table, there are 10 repositories and each of them includes a number of real and featured topics. For each repository, we compare its featured topics with the top-5 results retrieved by our approach. As it has been mentioned in Section 4.4, topics that are out of the training set from the real topics of a repository are removed. The last proposed topic is discovered by the language predictor component and the matched topics are printed in bold to distinguish them with the others. By all repositories, there is at least one matched item, i.e., success rate is equal to 1. Among others, given the testing project fishercoder1534/Leetcode, all the recommended topics are matched with those from the ground-truth data. As it can be seen, although the other recommended items do not belong to the ground-truth data, they are well correlated with the suggested ones. To be concrete, the topic *linux* is strongly related to *ubuntu* in the esell/deb-simple project although it does not appear in the real topics set. Given the circumstances, we suppose that the application of a lexical database to map words with similar semantic meaning such as WordNet [16] should possibly increase the overall prediction performance. However, this is out of scope of this paper, and thus we consider it as a future work.

5.2 Result Analysis

RQ$_1$: *How does the variation of training data impact on the prediction performance?*

Starting from the dataset described in Section 4.2, three different datasets have been populated, i.e., D$_1$, D$_2$, and D$_3$ by incrementally enlarging the considered training files as described in Table 2. We applied the same experimental settings described in Section 4.4 for all the datasets. Dataset D$_1$ consists of 1,340 README files and 10% of each topic are used as testing. Dataset D$_2$ and D$_3$ follow the same structure in which we increase the number of README files up to 50 and 100, respectively.

By running the tool on the datasets, for each testing item, we obtained a ranked list of topics that is considered to be relevant. Given the multi-classification problem, the number of recommended items dramatically affects the final results. As it has been shown in Figure 3, generally the repositories contain a low number of topics. Thus, we varied the number of retrieved topics for each testing item with a small value, i.e., $c = 2$, $c = 5$, and $c = 8$ recommended topics to find out the best configuration with respect to the returned items.

If N is the number of correctly predicted topics then for each setting, the corresponding quality metrics are computed with respect to N. The final success rates are depicted in Figure 5, Figure 6, and Figure 7 for $c = 2$, $c = 5$ and $c = 8$, respectively.

In Figure 5, it is evident that a better success rate is obtained when N=1, and this holds for all the datasets, i.e., D$_1$, D$_2$, and D$_3$. Running the tool on D$_2$ and D$_3$ yields a comparable success rate, i.e., 75.8% and 75.68%. This implies that at a certain point, increasing the amount of training data does not bring a radical change in the overall performance. This is interesting since it suggests that in practice, we just need a certain amount of training data in order to get a decent accuracy. When N=2, it can be seen that the obtained success

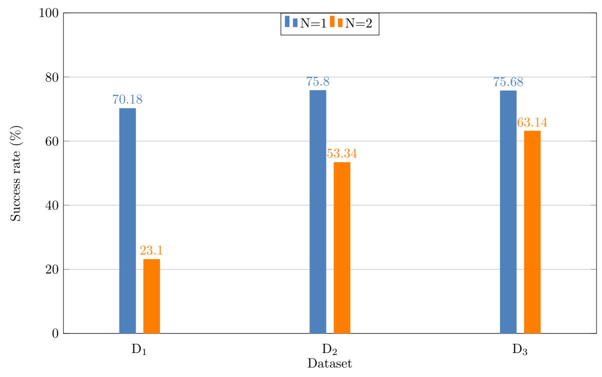


Figure 5. Success rate for c=2.

Table 1. Example of repositories, their topics and the recommended topics.

Repository	Real topics	Featured topics	Recommended topics
a1studmuffin/SpaceshipGenerator	python,blender-scripts, spaceship, procedural-generation, game-development, 3d	python, 3d	shell, terminal, **3d**, opengl, **python**
0xAX/go-algorithms	golang, algorithm, data-structures, go, sort, tree-structure	algorithm, data-structures, go	**data-structures**, **algorithm**, twitter, library, **go**
ajenti/ajenti	ajenti, python, javascript, administration, linux, panel, angular	python, javascript, linux, angular	firefox, webpack, **linux**, **angular**, **python**
fishercoder1534/Leetcode	leetcode, algorithm, java, interview, mysql, bash, apache, data-structures, leetcode-solutions, leetcode-questions, leetcode-java, leetcoder	algorithm, java, mysql, bash, data-structures	**algorithm**, **mysql**, **bash**, **data-structures**, **java**
cryogen-project/cryogen	clojure, cryogen, static-site-generator	clojure	jekyll, **clojure**, atom, gulp, html
alizahid/slinky	jquery, navigation, mobile, es6, javascript, menu, plugin, yarn, babel, webpack, css, sass	jquery, mobile, javascript, es6, css, sass, babel, webpack	**es6**, **jquery**, **css**, **sass**, **javascript**
esell/deb-simple	maintainer-wanted, golang, go, debian, ubuntu	go, ubuntu	**ubuntu**, ansible, linux, shell, **go**
JamesRamm/longclaw	python, django, e-commerce, shop, python3, python-2, wagtail, wagtail-cms	python, django, wagtail	**wagtail**, **django**, serverless, express, **python**
nitin42/terminal-in-react	terminal, react, design, javascript, svg, webpack, webapp, css	terminal, react, javascript, webpack, css	**webpack**, eslint, react-native, **react** , **javascript**
kerl/kerl	erlang, otp-release, kerl, homebrew, shell	erlang, homebrew, shell	elixir, **homebrew**, bash, deployment, **shell**

Table 2. Datasets.

Dataset	# of testing files	# of training files	Exe. time (sec)
D_1	134	1,206	658
D_2	670	6,030	3,782
D_3	1,340	12,060	7,823

rate improves considerably along the addition of more training data. For example, by D_1 we get 23.1% as success rate, however the corresponding value by D_3 is 63.14%, which is almost triple larger than that of D_1.

By considering Figure 6 and Figure 7 together to study the approach's performance for the cases c=5 and c=8, we witness the same trend as that by c=2. To be concrete, requesting more of correctly predicted topics means getting a lower success rate. For example when c=5, by Dataset D_1 the maximum success rate is 80.15% and it is obtained when N=1. This score decreases dramatically for a larger value of N, i.e., with N=4 corresponding the success rate is 19.83% which is four times lower than that of the case where N=1. This suggests that obtaining a higher number of correct topics becomes more difficult. Similarly, by D_2 and D_3, success rate deteriorates quickly if we want to see more matched topics. By comparing the results yielded by all three datasets, we see that running the MNB network on D_2 brings a slightly better prediction performance in terms of success rate, and this is consistent with what we got for c=2. This is further confirmed in Figure 7 as using D_2 for training data yields a better success rate.

These findings should be explained by the internal construction of MNB. As the network uses probabilities without

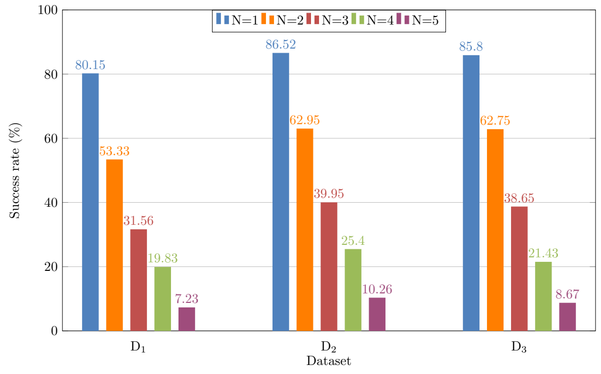


Figure 6. Success rate for c=5.

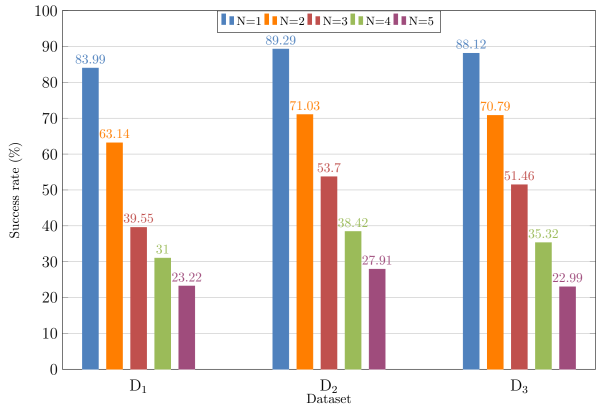


Figure 7. Success rate for c=8.

considering any other type of relationships among the features, it works well even with a small amount of data. Thus, the addition of further data has a negative impact on the learning rate of the MNB network. Concerning the number of recommended topics, our experimental configurations demonstrate that suggesting more topics increase the success rate. As a large part of the dataset contains repositories with at most 8 featured topics, those that have more than this number of topics are not statistically significant for our purposes. Additionally, we normalized success rate measures to be compliant with the distribution of the dataset.

Once a certain threshold has been reached, augmenting the training data does not bring a radical change in performance. In particular, adding more data to Dataset D_2 to yield Dataset D_3 has a negligible impact on success rate, precision, and recall.

RQ$_2$: *Is the approach able to provide consistent recommendations considering featured topics?*

Considering once again the three experimental datasets shown in Section 4.2, we computed precision and recall by varying the two values c and N. The precision, recall, and top-rank scores for all the test configurations are shown in Table 3. As it can be seen there, precision and recall scores decrease when N increases, which is consistent with success rate. Again, using D_2 as input data helps us achieve a better precision and recall compared to D_1 and D_3. Using D_3 helps obtain a better result with c=2. However, the difference between the two recall values is negligible. As expected considering the previous results, Dataset D_1 contributes to the worst performance along with all the possible configurations of N and c.

Table 3. Precision, recall, and top-rank.

	Precision			**Recall**			**Top rank**		
N	D_1	D_2	D_3	D_1	D_2	D_3	D_1	D_2	D_3
2	43.61	**47.98**	**47.98**	38.10	42.62	**42.96**	60.59	65.19	64.81
5	27.51	**31.20**	30.70	55.68	**63.45**	63.06	–	–	–
8	19.63	**22.05**	21.57	61.96	**69.77**	69.04	–	–	–

These results can be explained by the particular nature of the user's topics. As shown in Table 1, the featured topics belonging to the real repositories are very few. The side-effect of this assumption has a particular impact on the precision value, that goes down when we increase the number of recommended items. Meanwhile, recall does not suffer from the dataset composition, as this metric highlights the true positive ratio. We assume that the obtained results are strongly affected by the distribution of the user topics.

To further study the approach's performance, we computed Top Rank scores following Equation 7 and the results are depicted in Table 3. We considered only the first top rank items since all repositories belonging to the golden dataset have at least one featured topic. Thus, we exploit this index to assess the capability of the MNB network in a more reliable way. Once again, using D_2 brings a better top rank compared to using the other datasets. In particular, the Top Rank index increases to 65.19% from 60.59% when we run the tool on D_3 instead of D_1. Moreover, running the tool on D_3 places a burden on the overall performance. This confirms the findings of **RQ**$_2$, in which D_2 facilitates a better prediction.

Our approach is able to provide relevant results in terms of Top Rank scores given a decent amount of training data.

6 Threats to validity

We identify the threats that may adversely affect the validity of the evaluation, and the countermeasures taken to minimize them.

Threats to *internal validity* concern the criteria behind the selection of GitHub topics. As we already mentioned before, a GitHub's user can manually insert several numbers of topics. As this affects the quality of the final results, we considered only a limited set of featured topics. Another issue is related to the training phase that employs only README files to discover topics: They contain usually rough data that might not properly represent the entire repository's content. We mitigate this threat by obtaining a stable number of README files for each topic, and adding the GuessLang module to cover programming languages.

Concerning the threats to *external validity*, there might be issues with the selected dataset. First, we downloaded the projects directly from GitHub using the Python API. The platform limits the number of results to 1,000 for each query, so we were not able to access to the entire knowledge of the platform. Moreover, the query changes the set of the retrieved results at each run. Thus, the same query could retrieve different records into different runs. To avoid this situation, we imposed the above mentioned quality filter to cover projects as much as possible, without needing to execute a query several times.

Construct validity pertains to the conducted testing phase. As said, 10% of the README files for each topic are removed in every testing fold. In this way, we cannot test all possible permutation that might affect the training phase, as it requires the storage of a huge amount of data. To the best of our knowledge, the selected testing configuration is suitable to assess the approach's accuracy. Another issue concerns the chosen metrics to evaluate the approach: the overall precision considerably decreases due to the aleatory number of user's topics. We tackle this issue by setting the *Top rank* index to highlight the precision in terms of the first result.

7 Related Work

Orii [18] proposed a collaborative topic regression (CTR) model based on topics argued directly from the GitHub repository. The approach combines two different strategies to recommend a list of repositories given a pair user-repository. The first step involves a Gaussian model to compute matrix factorization. Through this strategy, it is possible to extract the latent vectors given a pre-computed matrix rating. Then, the approach employs a probabilistic topic modeling to discover topics from the repositories by analyzing high frequent terms. Finally, the two strategies were combined to build the CTR model. A dataset consisting of 120,867 repositories has been used to evaluate the proposed approach. After a manual classification of the repositories by examining source code, a five-fold cross validation was conducted to evaluate the approach by considering the pairs user-repository that have at least 3 watches.

SemiTagRec [8] is a semi-supervised approach aiming to recommend tags for Docker repositories. The tool is composed of four different components, i.e., predictor, extender, classifier, and integrator. Each of these components combines the results of the previous one. The predictor discovers an initial set of tags by using the unsupervised Latent Dirichlet Allocation (LDA) technique on the input repository. Then, the extender employs the learned tags to find out similar tags among the most popular topics provided by GitHub Repository Library. The third component implies a logistic regression model to argue new topics. This model is trained by a dataset manually labeled by the authors. As the model cannot cover all the topics due to the limited training data, the integrator aggregates the results obtained to deliver the final list of Docker tags.

Sharma *et al.* [22] exploit README files of GitHub projects to extract descriptive fragments in order to catalog them using the standard NLP techniques, i.e., tokenization, stop word removal, and stemming. The categorization has been done by using *LDA-GA*, a mature technique that combines Latent Dirichlet Allocation with Genetic Algorithm to build a topic model. This step returns a list of words representing the topic of the repository. A post-processing step was then manually conducted to remove the wrong terms or merge similar ones in terms of granularity. To evaluate the approach, a dataset composed of 10,000 GitHub repositories having at least 20 stars has been populated, and a user study has been done to assign a category to each repository. The experimental results show that the tool reaches an accuracy of 70%, and it is also capable of discovering new complementary categories.

Alreshedy *et al.* [3] proposed Source Code Classifier (SCC), a tool that can classify the language of snippets coming from StackOverflow posts. SCC uses the described Bayesian network to classify 20 different languages. The tool obtains 75% of precision, recall, and success rate. Our approach partially reuses SCC, which has been published under a free license. We extend the analysis of the network by considering README files coming from GitHub to recommend a set of topics instead of a single programming language class. Thus, we turn from a standard classification to a multi-classification problem using part of the same SCC infrastructure, i.e., the same MNB network and the TF-IDF vectorizer.

Developed for the InformatiCup 2017 competition, ClassifyHub [24] attempts to categorize GitHub repositories using a combination of eight weak classifiers. Each classifier works on different data, i.e., file extension, README files, commits, and repository structure. The final aim is to create a robust classifier using a combination of the weak ones, which perform slightly better than random classifiers. All the results of each classifier are combined to improve the learning phase. The final outcome of the tool is one of the seven categories manually defined by the authors. The approach has been evaluated using ten-fold cross validation over a dataset of 681 GitHub repositories. The results show that the combination of the weak classifier yields a precision and recall of 60%.

Sally [25] is a technique for automatic tagging of Maven-based software projects. The tool extracts identifiers and variables directly from bytecode. A filter is applied on the identifiers to produce primary and secondary tags for the analyzed projects. Primary tags are related to the project itself without considering dependencies, while secondary ones consider also dependencies. Moreover, Sally is able to build a dependency graph of the whole project by using the DepFind technique.[11] As the final output, Sally produces tag cloud to represent all the tags related to the project. This represents a visual hint for the user, as in the cloud the tags are classified by dimensions: the size is proportional related to the relevance for the project. The approach has been evaluated with the two most online tools for browsing dependencies in Java projects: SourceForge and MVNRepository. The experimental results demonstrate that Sally outperforms two existing approaches as it does not require additional information as categories or `pom.xml` files. The second experiment exploits MUDABlue [10] as baseline. A user study was conducted to evaluate the performance of both approaches. Also in this evaluation, Sally obtains a better performance compared to that of MUDABlue with respect to the expressiveness and completeness of the categorization task.

8 Conclusions

In this work, we have presented a novel approach to recommend a set of featured topics given a software project endowed with a corresponding README file. The tool is based on a probabilistic machine learning network, the Naïve Bayesian classifier. We encode the relevant information about repositories using the TF-IDF weight scheme. After the training phase, the approach provides the user with a list of featured topics related to its project. We evaluated the approach using cross-fold validation.

As future work, we plan to extend the analysis to source code of GitHub repositories to have more precise results in terms of accuracy. Moreover, we are going to integrate a reliable item-based collaborative filtering technique, with the aim of extending the final recommendation list. In particular, we attempt to solve the cold-start problem by using the MNB

[11] http://depfind.sourceforge.net/

network to discover an initial set of topics. It is our firm belief that a combination of these two complementary techniques will facilitate a better prediction.

References

[1] 2019. PyGithub/PyGithub. (Dec. 2019). https://github.com/PyGithub/PyGithub original-date: 2012-02-25T12:53:47Z.

[2] 2019. Understanding the search syntax - GitHub Help. (2019). https://help.github.com/en/github/searching-for-information-on-github/understanding-the-search-syntax

[3] Kamel Alreshedy, Dhanush Dharmaretnam, Daniel M. German, Venkatesh Srinivasan, and T. Aaron Gulliver. 2018. SCC: Automatic Classification of Code Snippets. *arXiv:1809.07945 [cs, stat]* (Sept. 2018). http://arxiv.org/abs/1809.07945 arXiv: 1809.07945.

[4] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Predicting the Popularity of GitHub Repositories. *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering - PROMISE 2016* (2016), 1–10. https://doi.org/10.1145/2972958.2972966 arXiv: 1607.04342.

[5] Hudson Borges, André C. Hora, and Marco Tulio Valente. 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*. 334–344. https://doi.org/10.1109/ICSME.2016.31

[6] Hudson Borges and Marco Tulio Valente. 2018. What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform. *Journal of Systems and Software* 146 (Dec. 2018), 112–129. https://doi.org/10.1016/j.jss.2018.09.016 arXiv: 1811.07643.

[7] Hudson Borges, Marco Tulio Valente, Andre Hora, and Jailton Coelho. 2017. On the Popularity of GitHub Applications: A Preliminary Note. *arXiv:1507.00604 [cs]* (March 2017). http://arxiv.org/abs/1507.00604 arXiv: 1507.00604.

[8] Wei Chen, Jia-Hong Zhou, Jia-Xin Zhu, Guo-Quan Wu, and Jun Wei. 2019. Semi-Supervised Learning Based Tag Recommendation for Docker Repositories. *Journal of Computer Science and Technology* 34, 5 (Sept. 2019), 957–971. https://doi.org/10.1007/s11390-019-1954-4

[9] Ganesan. 2019. Topic Suggestions for Millions of Repositories - The GitHub Blog. (2019). https://github.blog/2017-07-31-topics/

[10] Pankaj K. Garg, Shinji Kawaguchi, Makoto Matsushita, and Katsuro Inoue. 2004. MUDABlue: An Automatic Categorization System for Open Source Repositories. *2013 20th Asia-Pacific Software Engineering Conference (APSEC)* (2004), 184–193. https://doi.org/doi.ieeecomputersociety.org/10.1109/APSEC.2004.69

[11] Jing Jiang, David Lo, Jiahuan He, Xin Xia, Pavneet Singh Kochhar, and Li Zhang. 2017. Why and How Developers Fork What from Whom in GitHub. *Empirical Software Engineering* 22, 1 (Feb. 2017), 547–578. https://doi.org/10.1007/s10664-016-9436-6

[12] Liangxiao Jiang, Dianhong Wang, Zhihua Cai, and Xuesong Yan. 2007. Survey of Improving Naive Bayes for Classification. In *Advanced Data Mining and Applications*, Reda Alhajj, Hong Gao, Jianzhong Li, Xue Li, and Osmar R. Zaïane (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 134–145.

[13] Ashraf M. Kibriya, Eibe Frank, Bernhard Pfahringer, and Geoffrey Holmes. 2005. Multinomial Naive Bayes for Text Categorization Revisited. In *AI 2004: Advances in Artificial Intelligence*, Geoffrey I. Webb and Xinghuo Yu (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 488–499.

[14] Ron Kohavi. 1995. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2 (IJCAI'95)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1137–1143.

[15] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. 2007. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering* 160 (2007), 3–24.

[16] George A. Miller. 1995. WordNet: A Lexical Database for English. *Commun. ACM* 38, 11 (Nov. 1995), 39–41. https://doi.org/10.1145/219717.219748

[17] Phuong T. Nguyen, Juri Di Rocco, Davide Di Ruscio, and Massimiliano Di Penta. 2020. CrossRec: Supporting software developers by recommending third-party libraries. *Journal of Systems and Software* 161 (2020), 110460. https://doi.org/10.1016/j.jss.2019.110460

[18] Naoki Orii. 2013. Modeling for Recommending GitHub Repositories.

[19] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshynanyk, and A. De Lucia. 2013. How to effectively use topic models for software engineering tasks? An approach based on Genetic Algorithms. In *2013 35th International Conference on Software Engineering (ICSE)*. 522–531.

[20] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. 2019. Categorizing the Content of GitHub README Files. *Empirical Software Engineering* 24, 3 (2019), 1296–1327. https://doi.org/10.1007/s10664-018-9660-3

[21] Martin Robillard, Robert Walker, and Thomas Zimmermann. 2010. Recommendation Systems for Software Engineering. *IEEE Softw.* 27, 4 (July 2010), 80–86. https://doi.org/10.1109/MS.2009.161

[22] Abhishek Sharma, Ferdian Thung, Pavneet Singh Kochhar, Agus Sulistya, and David Lo. 2017. Cataloging GitHub Repositories. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE'17)*. ACM, New York, NY, USA, 314–319. https://doi.org/10.1145/3084226.3084287

[23] Claudio Di Sipio, Riccardo Rubei, Davide Di Ruscio, and Phuong T. Nguyen. 2020. A Multinomial Naïve Bayesian (MNB) network to automatically recommend topics for GitHub repositories - Online appendix. (February 2020). https://github.com/MDEGroup/MNB_TopicRecommendation/

[24] Marcus Soll and Malte Vosgerau. 2017. ClassifyHub: An Algorithm to Classify GitHub Repositories. In *KI 2017: Advances in Artificial Intelligence*, Gabriele Kern-Isberner, Johannes Fürnkranz, and Matthias Thimm (Eds.). Springer International Publishing, 373–379.

[25] S. Vargas-Baldrich, M. Linares-Vásquez, and D. Poshyvanyk. 2015. Automated Tagging of Software Projects Using Bytecode and Dependencies (N). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 289–294. https://doi.org/10.1109/ASE.2015.38