Topic Recommendation for GitHub Repositories: How Far Can Extreme Multi-Label Learning Go?

Ratnadira Widyasari, Zhipeng Zhao, Thanh Le Cong, Hong Jin Kang, David Lo
School of Computing and Information Systems, Singapore Management University
{ratnadiraw.2020, zpzhao, tlecong, hjkang.2018, davidlo}@smu.edu.sg

***Abstract*—GitHub is one of the most popular platforms for version control and collaboration. In GitHub, developers are able to assign related topics to their repositories, which is helpful for finding similar repositories. The topics that are assigned to repositories are varied and provide salient descriptions of the repository; some topics describe the technology employed in a project, while others describe functionality of the project, its goals, and its features. Topics are part of the metadata of a repository and are useful for the organization and discoverability of the repository. However, the number of topics is large and this makes it challenging to assign a relevant set of topics to a repository. While prior studies filter out infrequently occurring topics before their experiments, we find that these topics form the majority of the data.**

In this study, we try to address the problem of identifying the topics from a GitHub repository by treating it as an extreme multi-label learning (XML) problem. We collect data of 21K GitHub repositories containing 37K labels of topics. The main challenge for XML is a large number of possible labels and severe data sparsity which fit the challenge of identification of topics from the GitHub repository. We evaluate multiple XML techniques, such as Parabel, Bonsai, LightXML, and ZestXML. We then perform an analysis of the different models proposed for XML classification. The best results on all the metrics from XML models are from ZestXML which is a combination of zero-shot and XML. We also compare the performance of ZestXML with a baseline from a recent study. The results show that ZestXML improves the baseline in terms of the average F1-score by 17.35%. We also find that for the repositories that have topics that rarely appear in the repositories used during training, ZestXML improves the performance greatly. The average of F1-score is 3 times higher as compared to the baseline for the topics with 20 or less occurrences in training data.

***Index Terms*—multi-label classification, extreme multi-label learning, topic recommendation, GitHub repositories**

I. Introduction

Open source software (OSS) are often created through a joint-effort by a number of collaborating developers. Thus, a platform for developers to explore, share, and collaborate on OSS code is important. One of the most prominent OSS platforms is GitHub, where developers can code, manage, perform version control, and easily collaborate on the code and beyond [1]–[5]. To help other people find and contribute to related projects, in 2017, GitHub also provides developers with the capability to tag repositories with topics. Topics convey salient descriptions of the repository, communicating different details such as the goals of the repository, the features being developed, as well as technical details such as the libraries and frameworks employed in the project.

Since the introduction of topics in GitHub repositories, the overall number of available topics and the number of repositories using the topic features are increasing. Based on data that was collected on February 2020, Izadi et al. [6] highlighted that there are around two million repositories in GitHub that utilized the topic feature. By October 2022, the current number of repositories using that are tagged with at least one topic is around seven million repositories[1] . The number of default topics (i.e., topics that are officially listed by GitHub[2]) is also increasing. There were only 355 default topics in February 2020, while a total of 760 default topics are available in October 2022.

Topics are metadata of a repository that bridge the gap between its social and technical aspects [7]–[10]. A good assignment of topics benefit repositories by increasing their discoverability to potentially interested developers. Conversely, an incorrect assignment of topics may make it harder for the repository to be recommended to potential users. Therefore, correctly assigning appropriate topics to GitHub repositories is important. However, as the number of unique topics is large, solving this issue is challenging.

Researchers have attempted to automatically recommend topics for GitHub repositories based on their descriptions and README files [6], [11], [12]. Recently, Izadi et al. [6] attempted to address the aforementioned issue by treating the problem of assigning existing topics to repositories as a multi-label classification problem. However, a multi-label classification problem may not be effective when considering infrequent topics. In their experiments [6], infrequent topics with fewer than 100 occurrences were removed. After preprocessing the data, the number of topics in the dataset drops from 118K to 228. In a different study, topics occuring less than 20 times were remove, causing the number of topics to drop from 19,377 to 455 (only 3% of the original labels) [12] While reducing the overall number of topics makes it easier to assign the correct topics to a repository, it introduces some drawbacks. Reducing the number of topics may make it harder for developers to discover repositories for a newly emerging topic. As an example, consider a developer who is only interested in repositories related to a newly introduced version of a library (e.g., "react-router-v3"). By excluding the topics that have a low number of occurrences, newly

[1]Information retrieved using GitHub Advanced Search
[2]https://github.com/topics

emerging topics may not be considered. We later demonstrate that omitting topics with low frequency leads to a large loss of labels as the distribution of topics follows a long tail.

In this study, we collect a new dataset of GitHub repository topics. We collected a total of 37,161 unique topics from 21,273 GitHub repositories. As we do not filter out infrequent topics, our dataset has significantly more unique topics compared to the dataset used by Izadi et al. [6]. This results in a long-tail distribution where the majority of the topics have a low number of occurrences. A long tail significantly increases the challenge of identifying the correct topics for each repository as it contributes to data sparsity; there are numerous topics to choose from and each topic may only be associated with a small number of repositories. Moreover, each repository may be assigned more than one topic.

To address these challenges, we formulate the task of recommending topics for a GitHub repository as an XML (eXtreme Multi-Label Learning) problem. We investigate the effectiveness of XML techniques [13]–[23], which are machine learning approaches that are used for modelling data with extremely large numbers of labels. We also observe the need to address the challenge of predicting topics that occur only in the testing dataset but were previously unseen in the training dataset. To this end, we also experiment with ZestXML [24], a zero-shot XML technique.

In this study, we answer the following research questions:

1) **RQ-1: What is the impact of removing low frequency topics?** As the previous study preprocessed the dataset by removing low frequency topics, we investigate the effect of the removal of these topics on the dataset.
2) **RQ-2: Are XML methods effective in recommending topics of GitHub repositories?** We assess several XML methods (i.e., Parabel, Bonsai, LightXML, and ZestXML) as potential solutions for recommending topics from the GitHub repository.
3) **RQ-3: How does the performance of the best GitHub repository topic recommendation method identified in a prior study compare to Zero-shot XML?** We compare the best XML method, ZestXML, which is a zero-shot approach, to the best Multi-label classification technique identified in the previous study [6].
4) **RQ-4: How does the performance of the best method from the prior study compare to ZestXML in the low-frequency topic setting?** We compare the performance of ZestXML and Repologue using the data of low frequency topics, which would have been removed under the previous experimental setting.

We analyze if removing low-frequency topics affects our dataset. By removing the low-frequency topics with several thresholds (i.e., 20, 50, and 100) following previous studies [6], [11], [12], we lose a large proportion (over 95%) of topics. Overall, the total number of unique topics that are retained is less than 5% of the total topics. Meanwhile, the total number of topics of the GitHub repositories dropped from 158,054 to 60,496. These show that removing low-frequency topics affects our dataset significantly.

We use several XML models for our task of GitHub topic recommendation. Using our dataset, our experimental results show that Zero-shot XML (i.e., ZestXML) outperforms all the other XML models that we use for the experiment in all metrics that we use. We evaluate the tools using *precision*, *recall*, *F1-score*, and *success rate* calculated for the top K predicted topics, where the value of K varies. We find that a zero-shot XML approach provides the best results. This is indicates that there are many topics that were not present during training (i.e., only shows up on testing data). Indeed, there are 5,188 topics that show up only during testing.

Furthermore, we compare our best XML model, which is ZestXML to the state-of-the-art topic recommendation technique, Logistic Regression with TFIDF (LR-TFIDF) feature, which was used in the previous study [6]. We find that for all metrics at different top-k predictions, ZestXML outperforms LR-TFIDF. Calculating the average F1-score on all the considered top-k, we observe an improvement of 17.35%. In further analysis, we found that LR-TFIDF performance drops greatly on the topics that have low occurrences in training data. We found that for testing data where the ground truth topics only appear in less or equal than 5, 10, and 20 occurrences in training data, ZestXML outperforms LR-TFIDF by 9.5 times, 5 times, and 3 times respectively.

We perform a qualitative analysis to have a further understanding of the different results. From the analysis, we found that the LR-TFIDF approach pays more attention to the words that the model considers important for high-frequency topics. The number of unique topics that are used in the predictions from LR-TFIDF (3,057) is also small relative to ZestXML (11,801). This shows that LR-TFIDF is overly biased towards topics that occur frequently in the training data.

The main contributions of our work are as follows:

1) We investigate the effectiveness of several XML models to overcome challenges faced on the task of recommending topics to GitHub repositories. We found that ZestXML, which uses a zero-shot for XML, outperforms the other XML models and the method proposed in the previous study (i.e., Logistic Regression with TFIDF).
2) We perform a qualitative analysis to investigate the reasons for the performance improvement of the zero-shot XML approach compared to LR-TFIDF.
3) We show that the challenge of recommending topics to GitHub repositories is still an open challenge due to the difficulty presented by topics with low data frequency.
4) We make our datasets and code publicly available.

The rest of this paper is organized as follows. In Section II, we briefly describe the background of our study. In Section III, we discuss the XML models for topic recommendation. Section IV provides the experiment settings. We show the experiment results and answer the research questions in Section V. In Section VII, we discuss the threats to validity. Finally, we conclude our work and present future directions in Section VIII.

II. Background and Related Work

In this section, we introduce some brief background and related work: the first group of studies is about the problem of topic recommendation for the GitHub repository; the second group of studies is on extreme multi-label classification.

A. Topic Recommendation for GitHub Repository

Topic recommendation for GitHub repositories involves assigning topic tags to repositories in GitHub. Recent approaches for this problem can be categorized into two categories:

Probability-based Recommendation. The probability-based recommenders [11], [12], [25] employ *multi-class classification* algorithms, e.g., Multinomial Naïve Bayesian [11], to output the probability that a topic is relevant to a GitHub Repository. Then, they rank topics using the predicted probabilities to recommend topics. In this study, we do not include the probability-based recommendation techniques in our experiments as the replication package of its state-of-the-art technique, HybridRec [12] is incomplete at the time of writing this paper[3] while other approaches, e.g., MNBN [11], have been demonstrated to underperform multi-label-based approaches described below.

Multi-label Classification. Different from aforementioned approaches, Izadi et al. [6] recently formalized the task of topic recommendation for GitHub repositories as a *multi-label classification* (MLC) problem [26] and then conduct an empirical studies on several combination of (1) document representation and (2) multi-label classification. Their experiments showed that Logistic Regression with TF-IDF embeddings is the best combination for the topic recommendation, outperforming probability-based recommendation approaches [6].

As the number of labels of GitHub repositories can be expected to grow over time, existing techniques using a multi-label classification approach may not remain effective given the increasing scale. Indeed, the number of default topics on GitHub (i.e., the official topic list from the GitHub page) has grown from 355 in February 2020 to 776 in October 2022. Within our dataset, we find more than 37,000 topics. Moreover, we find that many topics have a low number of occurrences. These reasons motivate the use of other techniques that can address the increasingly large label space and the presence of low frequency data.

B. Extreme Multi-Label Learning

Extreme multi-label learning (XML) involves classifying documents with relevant labels from a huge label space. XML problems face the challenge of data sparsity, where each label may only have a small number of training examples for a model to learn from.

Supervised XML. The majority of extreme multi-label classifiers fall into the category of *supervised learning*. The approaches train one-vs-all [13]–[15], tree-based [16]–[19], embedding-based [20], [21] or deep-learning based [22], [23] classifiers on a standard training dataset in which every label contains multiple samples to assign labels for documents. Among these, Parabel [17], Bonsai [22] are notable for their good performance [22], [27]. Parabel and Bonsai are combinations of tree-based classifiers, which partition the label into a tree, and one-vs-all classifiers.Meanwhile, LightXML is a lightweight deep-learning model which fine-tunes transformer models for the task.

Zero-shot XML. Besides supervised learning approaches, ZestXML [24] has been proposed to use *zero-shot learning* for successfully predicting *zero-shot* labels which occur at test time but do not occur at training time. ZestXML have been demonstrated to be more accurate than supervised learning approaches on the zero-shot settings [24], [28]. In Software Engineering research, some previous studies [27], [28] have applied XML models for a different task from this study where it is used to identify libraries from vulnerability reports.

III. XML Models for Topic Recommendation

In this section, we first reformulate topic recommendation as an XML problem and then briefly introduce the four XML models used for this task: Parabel, Bonsai, LightXML, and ZestXML. We selected these models since they demonstrated strong performance on XML tasks [22], [24], [27].

A. Problem Formulation

Prior Knowledge. A labelled dataset $\mathcal{D} = (\mathcal{G}, \mathcal{T}, \mathcal{M})$, where $\mathcal{G}$ is set of GitHub repositories and $\mathcal{T}$ is set of topics. $\mathcal{M}$ is a mapping from $\mathcal{G}$ to set of subsets of $\mathcal{T}$, where $\mathcal{M}(g) \subseteq \mathcal{T}$ is the set of topics for a GitHub repository g. $|T|$ is extremely large.

Input. A new (unlabelled) dataset $\mathcal{D}_{new} = (\mathcal{G}_{new}, \mathcal{T}_{new})$ where $\mathcal{G}_{new} \neq \mathcal{G}$ is a set of repositories and $\mathcal{T}_{new} \supseteq \mathcal{T}$ is set of labels. We consider the repositories' descriptions and README files in this study.

Output. A mapping $\mathcal{M}_{new}$ from $\mathcal{G}_{new}$ to set of subsets of $\mathcal{T}_{new}$ such that $\mathcal{M}_{new}(g) \subseteq \mathcal{T}_{new}$ is the set of labels for each repository $g \in \mathcal{G}_{new}$.

B. Parabel

Parabel is a combination of a tree-based XML approach and a one-vs-all XML approach. Parabel [29] first leverages tree-based classifiers to learn up to three label trees by recursively partitioning the labels into two balanced groups using balanced 2-means clustering. Each non-leaf node in these label trees is then assigned a binary classifier to decide whether a document should be passed down to the left, right, or both child nodes. As a document may arrive into multiple leaf nodes, Parabel finally employs several linear one-vs-all classifiers, one for each label contained in the leaf, to predict the corresponding labels. By combining both tree-based and one-vs-all approaches, Parabel cuts down the cost of training and prediction (by reducing label space via tree-based classifiers) while maintaining a high prediction accuracy (via one-vs-all classifiers).

[3]We have discussed this with the authors. They informed us that they are currently working on the README and a running example of the tool.

C. Bonsai

Bonsai [30] uses the same learning paradigm as Parabel. Different from Parabel, Bonsai does not construct deep and balanced trees to avoid error propagation due to cascading effect of the deep trees [30]. Instead, the approach creates a diverse and shallow tree by using K-means clustering with a large K, e.g., K=100, in the implementation and does not balance the tree.

D. LightXML

LightXML [22] is a deep learning model, which fine-tunes a single transformer model with dynamic negative label sampling. LightXML model consists of four parts: (1) label clustering, (2) document representation, (3) label recalling, and (4) label ranking. Firstly, LightXML employs a balanced 2-means clustering to partition the labels into clusters such that each label belongs to one label cluster. LightXML then uses Transformer models to embed documents into a high-dimensional numerical vector. For this purpose, three pre-trained transformer models are used: BERT [31], XLNet [32], and RoBERTa [33]. Finally, given label clusters and document representations, LightXML leverage generative cooperative networks with dynamic negative label sampling to recall and rank labels. Particularly, for recalling labels, LightXML uses the generator network scores all label clusters and returns potential labels. Then, LightXML uses the discriminator network to distinguish correct labels from potential labels.

E. ZestXML

ZestXML [24] is a recently proposed approach for zero-shot XML where relevant labels have to be chosen from all available seen and unseen labels. Toward this, ZestXML first uses TF-IDF [34] to extract the feature vectors for the descriptions and labels. ZestXML then learns to project a document's features close to the features of its relevant labels through a highly sparsified linear transform and a novel second-order optimizer. ZestXML proposed a novel second-order optimizer called XHTP. Through leveraging assumptions such as feature independence and a favorable starting point, XHTP is able to efficiently produce highly sparse and accurate model parameters in a single iteration of approximation and refinement. ZestXML has been demonstrated to be more accurate than supervised learning approaches on the zero-shot experimental setting [24].

IV. Experiment Settings

A. Dataset

Since the raw data collected in the prior works [6], [12] is not available[4] and the publicly available dataset of processed data does not fit with our problem setting (as they do not retain low frequency labels as explained in Section I), we collected our own dataset from GitHubby using its REST API[5].

[4]Through private communications, the authors informed us that they did not keep the raw data.

[5]https://docs.github.com/en/rest

TABLE I: The statistics of training, validation and testing dataset

Dataset	#GitHub Resipotories	#Topics
Training	17,018	31,973
Testing	4,255	12,912
Total	21,273	37,161

Particularly, we implement a script to automatically retrieve data from GitHub repositories. Each repository is represented by a document containing the repository's description and README [35]. We also obtain the ground-truth topics of each repository, including user-defined topics. For the collection of the repositories, we only consider GitHub repositories with at least 1,000 stars.

Before using the dataset, we perform the preprocessing steps following Izadi et al. [6]. Note that we skip some preprocessing steps to retain the diversity of the topic space. For example, we do not remove the digits from the topics as we consider `python2` and `python3` as different topics. We first remove repositories with no README, description, or topic. This results in 22,161 GitHub repositories and 38,181 unique user-defined topics. Then, we perform some preprocessing steps on user-defined topics and repositories' documents as described below.

Topics. To clean user-defined topics, we first convert plural forms to singular, e.g., `networks` is converted to `network`. Then, similar to document preprocessing, we also replace common Computer Science abbreviations with their original words. Finally, we merge a set of related topics to a larger concept by using the mapping provided by Izadi et al. [6]. For example, we tag a topic `neural-network` to repositories with both neural and network topics.

Documents. To clean documents, we remove non-alphanumeric characters by using regular expressions and replace common Computer Science abbreviations with their original words, e.g., `os` is expanded to `operating-system`. The details of abbreviations and regular expressions are given in our replication package. Finally, following Izadi et al. [6], we exclude repositories in which more than half of their documents contain non-English characters.

After these preprocessing steps, we have a final dataset of 21,273 GitHub repositories with 37,161 unique topics. For a fair comparison between the models, we use the same training and testing dataset. Particularly, we randomly split the dataset with an 80:20 ratio, where the 80% forms the training dataset while the remaining forms the test dataset. The statistics of our training and testing datasets are shown in Table I.

B. Evaluation Metrics

In this study, we evaluate the performance of the models through four different metrics, which are precision (P), recall (R), F1-score (F1), and success rate (SR) These metrics have been widely used in experiments on multi-label classification and topic recommendation systems [6], [12], [36], [37]. We

calculate these metrics while using a varying number of the top *K* prediction results of each model. We choose 3 values of *K* which are 5, 8, and 10, since the average number of topics tagged to the repositories in our dataset is 7.5. Most of the repositories (i.e., 70%) in our dataset have at least 5 topics. Meanwhile, the percentage of our repositories that have at least 10 topics is 25%. These distribution and statistics are inline with both the testing and training data. Considering the different top *K* values, in total, we use 12 different metrics in this study, which are P@5, R@5, F1@5, SR@5, P@8, R@8, F1@8, SR@8, and so on.

Precision, Recall, and F1. Given the top-K predicted topics for a repository r, $pd\_k(r)$, and the actual labels $lb(r)$ for a corresponding repository, the $P@k(r)$ and $R@k(r)$ are defined as follows:

$$P@k(r) = \frac{pd\_k(r) \cap lb(r)}{k} \quad R@k(r) = \frac{pd\_k(r) \cap lb(r)}{|lb(r)|}$$

After we obtain the precision and recall for each repository, we compute the average of $P@k$ and $R@k$. Given N as the total number of repositories, $P@k$ and $R@k$ are defined as follows:

$$P@k = \frac{1}{N}\sum_{r=1}^{N} P@k(r) \quad R@k = \frac{1}{N}\sum_{r=1}^{N} R@k(r)$$

For the $F@k$ calculation, we compute the harmonic mean of $P@k$ and $R@k$. $F@k$ is defined as follows:

$$F@k = 2 \times \frac{P@k \times R@k}{P@k + R@k}$$

Success Rate. Following previous studies [6], [12], the success rate (SR) metric is defined as the percentage of repositories where the predictions from the model have at least one topic that matches with the actual topic. Similar to the other metrics, we consider different top K prediction values. For example, $SR@5$ measures the percentage of repositories that have at least one matching prediction from the top-5 predictions with the actual label. Note that Success Rate is a *weaker* evaluation metric than F1 as it only checks for occurrence of only one out of the possibly many topics that can be assigned to a repository in a recommendation. The formula for success rate is defined as follows, with N as the total number of repositories:

$$SR@k(r) = |pd\_k(r) \cap lb(r)| > 0$$

$$SR@k = \frac{1}{N}\sum_{r=1}^{N} SR@k(r)$$

Note that Success Rate is a *weaker* evaluation metrics than F1 as it only checks for occurrence of one out of the possibly many topics that can be assigned to a repository in a recommendation.

TABLE II: Parameters used to train Parabel and Bonsai

Parameter	Parabel	Bonsai
Clustering function	Balanced 2-means	K-means
Cluster size	3	100
Num trees	3	3
Loss function	hinge	hinge
Maximum depth	20	20
Feature	TF-IDF	TF-IDF

TABLE III: Parameters used to train LightXML

Parameter	Value
Learning rate	0.00001
Epoch	20
Batch size	8
SWA warmup	10
SWA step	200
Feature	Transformer generated vectors

C. Implementation Details

1) Implementation and Enviroment: For XML models, we implement the proposed approaches using the PyTorch library and Python programming language. The models are trained and evaluated on a Docker environment running Ubuntu 18.04 with Intel(R) i7-10700K @3.8GHz, 64GB RAM, and 2 NVIDIA RTX 2080Ti GPU (11GB of graphics memory for each). We utilize the CPU for training all models except LightXML, which require the GPU for training. For LR-TFIDF, we reuse the code in their replication package [38] where they use Python library scikit-learn [39] to implement LR and TFIDF. The model is trained and evaluated on a Docker environment running Ubuntu 18.04 with Intel(R) Xeon(R) CPU E5-2698 v4 @2.2GHz, 503GB RAM.

2) Model Parameter Setting: In our experiments, we use the same set of model parameters with their original works [6], [16], [22], [24], [29] except the batch size of LightXML. Particularly, we use a smaller batch size as the total number of topics considered in our dataset significantly increased compared to the previous study. As the total number of topics increases, the required GPU memory to model all the possible topics also increased, resulting in the decreasing batch size that can be used to train the model. This is a general problem that is faced by all researchers on this task. The detailed parameters of Parabel and Bonsai, LightXML, ZestXML, and LR-TFIDF are shown in Table II, III, IV, V, respectively.

3) Experimental Setup: With respect to the training and testing procedure, the performance of the model may fluctuate slightly due to the randomness in the learning algorithm. To mitigate the effect of randomness on our evaluation, following the prior work [27] we run the experiment ten times for Parabel and Bonsai and then report the average of the performance metrics. For others, we construct one model each as the performances of these models do not fluctuate.

D. Research Questions

1) RQ-1: What is the impact of including low frequency topics in our dataset?

In this RQ, we want to analyze the data that we collected as described in Section IV-A. Previous papers [6], [12] in this

TABLE IV: Parameters used to train ZestXML

Parameter	Value
Model Sparsity K	10
Shortlist Size S	100
Common Regulation Parameter	1

TABLE V: Parameters used to train LR-TFIDF

Parameter	Value
Embedding Size	20000
N-grams	1,2
Low Frequency Threshold	50

topic propose several preprocessing approaches, including the removal of unrepresentative topics from the dataset. This is done to ensure that each topic in the dataset has a sufficiently large number of occurrences, which is important for machine learning approaches to work effectively. In the study by Izadi et al. [6], the authors removed topics with fewer than 100 occurrences. Meanwhile, Di Rocco et al. [12] removed topics occuring fewer than 50 times and 20 times for their experiment. The act of removing less frequent topics may remove a large proportion of topics, therefore, we investigate the effect of the removing less frequent topics.

2) RQ-2: Are XML methods effective in GitHub repositories topic recommendation?

Next, we investigate the effectiveness of several XML models (i.e., Bonsai, Parabel, LightXML, and ZestXML) for our problem of topic recommendation for GitHub repositories. We formulate the problem as an XML task as it shares characteristics with XML problems. In particular, there is a large number of labels with severe data sparsity as many labels may appear infrequently. We aim to discover which XML model is most effective for the recommending topics for GitHub repositories. To evaluate a model's performance, we use several metrics, including precision, recall, F1 score, and success rate. For each metric, we also consider different numbers of the model's top predictions (i.e., top-5, top-8, and top-10 predictions).

3) RQ-3: How does the performance of the previous method compare to ZestXML?

We perform experiments involving the best model found by Izadi et al. [6], which uses an LR (Logistic Regression) model after using TFIDF. We run the LR-TFIDF model based on the code provided from the previous study by Izadi et al. [6].Then, we compare it against the best approach we found in RQ-2. We investigate the performance of this LR-TFIDF model. Similar to the previous research question (i.e., RQ-2), we use four different performance metrics while considering a varying number of the top predictions by each model.

4) RQ-4: How does the performance of LR-TFIDF compare to ZestXML in the low-frequency topic setting?

We investigate the performance of the techniques on low-frequency topics. We compare the results of LR-TFIDF compared to ZestXML on the repositories where all the topics only show up a few times in the training data. We use the same performance metrics as RQ-2 and RQ-3. We use several thresholds for filtering the low-frequency topics data which are 5, 10, and 20 for the analysis.

V. Results

A. RQ-1 What is the impact of including low frequency topics in our dataset?

As discussed in Section IV-A, our dataset consists of 21,273 GitHub repositories with 37,161 unique topics. The frequency distribution of topics in our dataset is shown in Figure 1. We observe that the frequency of topics follows a long-tail distribution. A long-tail distribution has the characteristic of a large number of labels that occurs at a low frequency, creating a long "tail" that gradually tapers off towards the end of the distribution [40]. Data from problems with such a characteristic may not be easily modeled using only imbalanced classification approaches. Thus, few-shot learning and even zero-shot learning approaches are often proposed to address the challenge of long-tail distribution.

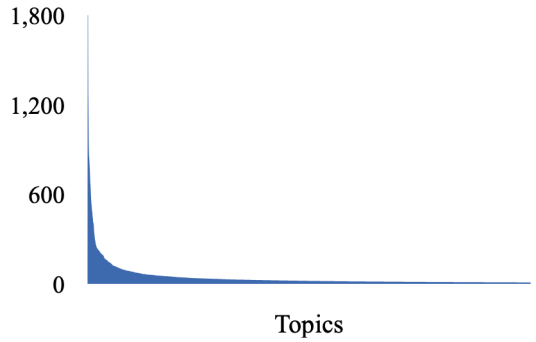


Fig. 1: The distribution of topic frequency which follows a long tail. 11.5% of topics appear less than 5 times.

We investigate the impact of removing less frequent topics (i.e., topics with a small number of occurrences) used as a pre-processing step used in the previous studies [6], [12]. We experiment with several frequency thresholds for removing topics. Specifically, we consider the following thresholds:

1) less than 100 occurrences;
2) less than 50 occurrences;
3) less than 20 occurrences.

After removing the topics that have less than 100 occurrences, the dataset is left with a total of 154 topics out of 31,761 topics, or just 0.5% of the original number of topics. After removing the topics that occur less than 50 times, we are left with 382 topics, or just 1.2% of the original topics. After removing the topics that occur less than 20 times, we are left with 1,108 topics, or just 3% of the original topics.

The total number of occurrences of topics that are removed using threshold 100 is 114,497, which is 72% from the total number of topics usages (i.e., 158,054). Meanwhile, for threshold 50, the total number of removed topic occurrences is 99,391 (63%). For threshold 20, the total number of occurrences of the topics that are removed is 78,786 (50%).

Removing less frequent topics impacts on the number of topics of each repository. After removing topics with less than

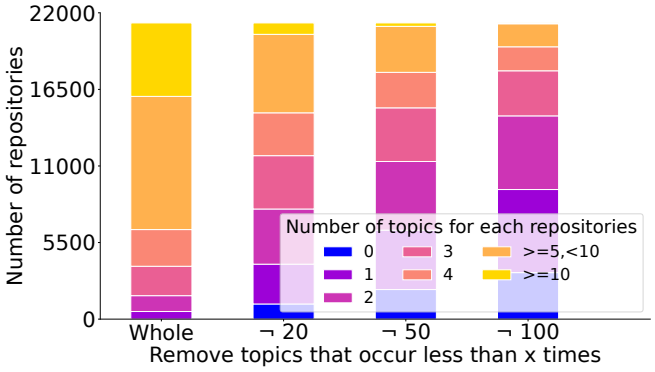


Fig. 2: Number of topics in repositories before and after removal of topics

100 occurrences, the average number of topics per repository dropped from 7.5 to 2. The median number of topics dropped from 6 to 2. There are 3,340 repositories (16%) out of 21,273 that no longer have topics. Before removing topics, 70% the repositories have at least 5 topics; however, after the removing topics, most of the repositories (92%) has less than 5 topics. These dropped numbers of average and median are also inline in the removal of topics that are less than 50 and 20 occurrences. The average of topics per repository after removing the topics that occur less than 50 and 20 times are 2.8 and 3.7 respectively. Meanwhile, the median of topics per repository after removing the topics that occur less than 50 and 20 times are 2 and 3 respectively. The number of repositories that no longer have topics after the removal of less frequent topics occurring less than 50 and 20 times are 2,125 and 1,088 respectively. After removing the less frequent topics occurring less than 50 and 20 times, most repositories (i.e., 83% and 70%) have less than 5 topics. Figure 2 shows the distribution of topics within repositories before and after the removal process.

In summary, the removal of less frequent topics leads to the loss of over 95% of total topics. Since a majority of topics were removed, many of these topics were likely to be of interest to GitHub users. They may be important for the popularity and discoverability of a repository on GitHub. Hence, we evaluate the topic recommendation approaches when infrequent topics are retained in the dataset. In this study, we do not filter out the less frequent topics from our experiments.

Answer to RQ1: As the topic frequency has a long-tail distribution, the removal of less frequent topics leads to the loss of over 95% of topics, which may contain important and newly-emerging topics.

B. RQ-2 Are XML methods effective in recommending topics for GitHub repositories?

As discussed in Section IV-A, the dataset we employed has a large number of topics (i.e., 31,761 topics, compared to the previous dataset with only 228 topics). In RQ-1, we observe data sparsity from the high number of low-frequency topics. From these characteristics, we believe that the task of recommending topics is an extreme multi-label learning (XML) problem. To investigate the effectiveness of XML methods, we experiments with several methods. These methods are Bonsai, Parabel, LightXML, and ZestXML, described in Section II.

Our experimental results are shown in Table VI. Overall, ZestXML performs the best. ZestXML outperforms the other XML models on every evaluation metric. In terms of average F1, the improvements of ZestXML over Bonsai, Parabel, and LightXML are 21.7%, 26.5%, and 25.6% respectively. For the top-5 prediction results, ZestXML outperforms Bonsai, Parabel, and LightXML in terms of F1 scores by 19.5%, 23.9%, and 22.7% respectively. For the top-8 prediction results, the F1-score improvement of ZestXML compared to Bonsai, Parabel, and LightXML models are 22.9%, 27.8%, and 27.1% respectively. For the top-10 prediction results, the F1-score improvement of ZestXML compared to Bonsai, Parabel, and LightXML models are 24.2%, 28.9%, and 28.4% respectively. As the number of predictions increase, ZestXML has greater improvement over the existing approaches.

Furthermore, for the success rate in the top-5 prediction results, ZestXML achieves an improvement of 7.7%, 9.3%, and 7.3% compared to Bonsai, Parabel, and LightXML respectively. This improvement in the top-5 results by ZestXML is the biggest improvement in terms of success rate compare to top-8 and top-10 results. For the top-8 prediction results, the success rate improvement of ZestXML compare to Bonsai, Parabel, and LightXML models are 6.7%, 8.1%, and 6.4% respectively. For the top-10 prediction results, the F1-score improvement of ZestXML compare to Bonsai, Parabel, and LightXML models are 6.5%, 7.7%, and 6.3% respectively.

These results show that ZestXML is more effective than the other models. This may be caused by a large number of labels that did not occur in the training dataset but appeared in the testing data. 5,188 out of 31,761 labels occur only in the testing data. For all repositories with these labels in the training, the models other than ZestXML incorrectly predicted other labels. This shows that considering zero-shot learning is important in our study. For the remaining RQs (RQ-3 and RQ-4), we use ZestXML, which is the best-performing XML model based on these experimental results.

Answer to RQ2: ZestXML is the best-performing XML model on topic recommendation for GitHub repositories with an average success rate of 0.95 and an average F1-score of 0.39.

C. RQ-3 How does the performance of the best non-XML method compare to ZestXML?

In RQ-2, we have compared the performance of multiple XML models in GitHub topic recommendation, identifying ZestXML as the best-performing XML model. In RQ-3, we compare the performance of the method proposed in the previous study, LR-TFIDF [6] (i.e., the best-performing model in their study) with ZestXML.

TABLE VI: Performance of XML models in several metrics. Average F1 is calculated by taking the average of F1@5, F1@8, and F1@10. Meanwhile, average SR is calculated by taking the average of SR@5, SR@8, and SR@10. Results in **bold** indicate the best performance of a metric.

Model	P@5	R@5	F@5	SR@5	P@8	R@8	F@8	SR@8	P@10	R@10	F@10	SR@10	Avg. F1	Avg. SR
Parabel	0.35	0.29	0.32	0.85	0.27	0.35	0.30	0.88	0.23	0.37	0.29	0.90	0.31	0.88
Bonsai	0.37	0.30	0.33	0.86	0.28	0.36	0.32	0.90	0.24	0.39	0.30	0.91	0.32	0.89
LightXML	0.36	0.30	0.32	0.87	0.27	0.35	0.31	0.90	0.24	0.37	0.29	0.91	0.31	0.89
ZestXML	**0.43**	**0.37**	**0.40**	**0.93**	**0.34**	**0.45**	**0.39**	**0.96**	**0.30**	**0.48**	**0.37**	**0.97**	**0.39**	**0.95**

Our experimental results of LR-TFIDF against ZestXML is shown in Table VII. ZestXML achieves better results than LR-TFIDF on all metrics. The largest improvement in the terms of F1 is from the top-5 prediction results, where ZestXML outperforms LR-TFIDF by 23.4%. For the top-8 and top-10 prediction results, the improvements made by ZestXML are 14.5% and 15.4% respectively. Overall, the improvement of average F1 from ZestXML is 17.35%. Compared to LR-TFIDF, ZestXML has a higher success rate with an average improvement of 5.2%. For the top-5, top-8, and top-10, the improvements of ZestXML are 5.9%, 4.8%, and 4.8%.

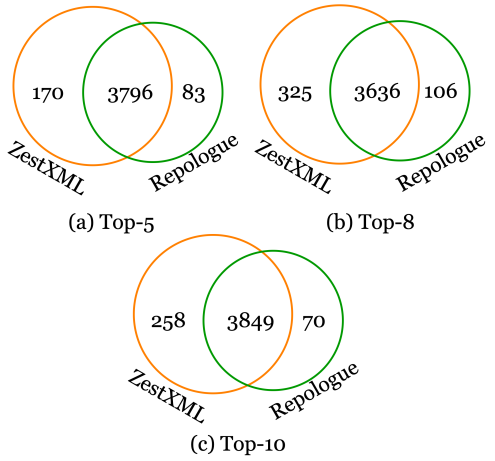


Fig. 3: Venn diagram of repositories with a corresponding topic that is successfully identified in the top-5 predictions by ZestXML and LR-TFIDF

Figure 3a shows the number of repositories that were successfully identified by ZestXML and LR-TFIDF within the top-5 predictions. From 3,742 repositories that are successfully identified in top-5 by LR-TFIDF, there are 3,636 repositories (97%) identified in the top-5 predictions of ZestXML. For the other 106 repositories with labels that are identified only in the top-5 predictions by LR, more than half (i.e., 58%) were identified within the top-10 predictions of ZestXML. Meanwhile, from 3,961 repositories with topics identified in the top-5 predictions of ZestXML, LR-TFIDF fails to identify 219 repositories within its top-5 predictions. The statistics of the diagram are also inline in the top-8 and top-10 prediction results which can be seen in Figure 3b and Figure 3c respectively. The ratio of repositories number that can only be identified in ZestXML compared to LR-TFIDF in the top-5 is 75:25. While on top-8 and top-10, the ratios are 76:24 and 79:21 respectively.

Answer to RQ3: ZestXML improves over the state-of-the-art non-XML baseline by 5.2% and 17.35% in terms of average success rate and F1-score, respectively.

D. RQ-4 How does the performance of LR-TFIDF compare to ZestXML in the low frequency topic setting?

We further analyze the performance of the approaches on low-frequency topics. For low-frequency topics, LR-TFIDF performs worse than ZestXML. We filter the repositories, selecting repositories where all of their actual labels in the testing data have low occurrences in the training data. To identify low frequency topics, we set several frequency thresholds which are 5, 10, and 20. For example, a threshold of 5 means that only repositories in testing where all their actual corresponding labels have a frequency less than or equal to 5 in the training data. After filtering using a threshold of 5, the number of remaining repositories in the testing data is 112 out of 21,273 (0.5%). For thresholds 10 and 20, the number of remaining repositories in the testing dataset are 162 and 262 respectively.

The results of both models on low-frequency topics are shown in Table VIII. Using a threshold of 5, based on average F1, ZestXML outperforms LR-TFIDF by 9.5 times. In terms of F1@5, F1@8, and F1@10, ZestXML is more effective than LR-TFIDF by 7 times, 9 times, and 8.5 times respectively. For the average success rate, the result of ZestXML is 7 times the result of LR-TFIDF. For the SR@5, SR@8, and SR@10, the value of ZestXML are 8 times, 6 times, and 6.5 times from the LR-TFIDF result respectively. These improvements are also observed when using a threshold of 10 and 20. The average F1-scores of ZestXML in the threshold of 10 and 20 are 5 times and 3 times of LR-TFIDF respectively. The lower the frequency of topics that occur in the training data, the greater the difference between ZestXML and LR-TFIDF. This highlights the strength of ZestXML when considering less frequent topics, which reflects a more practical setting as the majority of labels are infrequent.

Answer to RQ4: On the low frequency topic setting (threshold 20), ZestXML outperforms the baseline, LR-TFIDF by 2 and 3 times in terms of average success rate and F1-score, respectively. This result is in line with the lower threshold (i.e., 5 and 10).

TABLE VII: Performance of ZestXML and LR-TFIDF in several metrics. Results in **bold** indicate the best performing approach on a metric

Models	P@5	R@5	F1@5	SR@5	P@8	R@8	F1@8	SR@8	P@10	R@10	F1@10	SR@10	Avg. F1	Avg. SR
LR-TFIDF	0.39	0.32	0.32	0.88	0.30	0.39	0.34	0.91	0.26	0.41	0.32	0.92	0.33	0.90
ZestXML	**0.43**	**0.37**	**0.40**	**0.93**	**0.34**	**0.45**	**0.39**	**0.96**	**0.30**	**0.48**	**0.37**	**0.97**	**0.39**	**0.95**

TABLE VIII: Performance of ZestXML and LR-TFIDF on the low frequency topics. Results in **bold** shows the best performing approach for a metric

Threshold: 5														
Model	P@5	R@5	F1@5	SR@5	P@8	R@8	F1@8	SR@8	P@10	R@10	F1@10	SR@10	Avg. F1	Avg. SR
LR-TFIDF	0.02	0.04	0.03	0.07	0.01	0.06	0.02	0.10	0.08	0.01	0.02	0.11	0.02	0.09
ZestXML	**0.14**	**0.38**	**0.20**	**0.55**	**0.11**	**0.47**	**0.18**	**0.63**	**0.10**	**0.54**	**0.17**	**0.71**	**0.19**	**0.63**
Threshold: 10														
Model	P@5	R@5	F1@5	SR@5	P@8	R@8	F1@8	SR@8	P@10	R@10	F1@10	SR@10	Avg. F1	Avg. SR
LR-TFIDF	0.03	0.06	0.04	0.12	0.02	0.09	0.04	0.17	0.02	0.10	0.03	0.18	0.04	0.15
ZestXML	**0.15**	**0.36**	**0.21**	**0.56**	**0.12**	**0.45**	**0.19**	**0.65**	**0.11**	**0.51**	**0.19**	**0.72**	**0.20**	**0.64**
Threshold: 20														
Model	P@5	R@5	F1@5	SR@5	P@8	R@8	F1@8	SR@8	P@10	R@10	F1@10	SR@10	Avg. F1	Avg. SR
LR-TFIDF	0.07	0.12	0.08	0.25	0.06	0.17	0.08	0.33	0.05	0.18	0.08	0.36	0.08	0.32
ZestXML	**0.17**	**0.36**	**0.23**	**0.61**	**0.14**	**0.45**	**0.22**	**0.70**	**0.13**	**0.51**	**0.21**	**0.76**	**0.22**	**0.69**

VI. Discussion

High success rate. The majority of repositories have at least one high frequency topic. 88% (3,735 out of 4,256) of the repositories in the testing dataset have topics that occur more than 50 times during training. As the success rate metric considers a recommendation successful as long as just one ground-truth topic matches the recommendation (as described in Section IV-B), LR-TFIDF seems to performs well on these metrics although it underperforms ZestXML on the other metrics. Moreover, our detailed experiments indicate that LR-TFIDF has poor performance on low frequency topics. On both the entire dataset and the subset of the data with low frequency topics, ZestXML outperforms LF-TFIDF. Hence, we focus our qualitative analysis on the repositories whose topics occur with low frequency.

Qualitative Analysis To have a better understanding of the experimental results, we perform a qualitative analysis on ZestXML and LR-TFIDF by analysing the explanations of each prediction. To generate an explanation of ZestXML, we obtain the feature relevance scores, produced by ZestXML for the documents' text and labels. Using the feature importance scores, we obtain an explanation for every prediction made by the ZestXML model. Then, we utilize Pylighter[6] to highlight the text, with a stronger highlight if the score is higher.

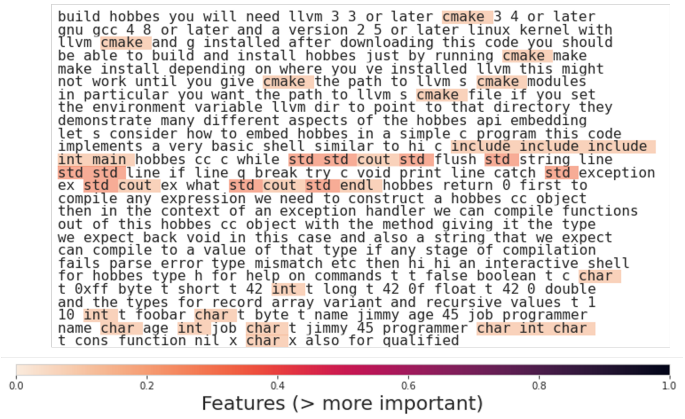
For the LR-TFIDF model, we utilize LIME (Local Interpretable Model-Agnostic Explanation) [41] to get the scores of how the feature (i.e., word token) contributes to the model prediction and visualize it. LIME is one of the most popular techniques in eXplainabe Artificial Intelligence (XAI). The explanation generated for LR-TFIDF is also consistent with the explanation from ZestXML, the highlighted texts are important parts for the model to make a decision.

For the qualitative analysis, we randomly inspect the feature importance scores obtained from LR-TFIDF and ZestXML for low-frequency topics appearing in less than five repositories. We observe that the LR-TFIDF puts more weight on words that the model considers important for predicting *high-frequency topics*. Take Figure 4a as an example. LR-TFIDF focuses on words such as "std" and "cmake", which are related to a *high-frequency topic*, i.e. cpp, while ignoring important words from *low-frequency topics*, such as "hobbes" even when these words are related to the correct predictions. As a consequence, LR-TFIDF fails to predict a correct topic among its top-10 predictions. The observation shows that LR-TFIDF is biased by high-frequency topics when modeling the input text. Meanwhile, as seen in Figure 4b, ZestXML succeeds in giving importance to words related to low-frequency topics, and so, ZestXML correctly predicts "hobbes" as a relevant topic.

Furthermore, we observe that LR-TFIDF does not give enough weight to important words that are relevant only to low-frequency topics. Indeed, when a repository contains only words relevant to low-frequency topics, LR-TFIDF produces only predictions with low probabilities. Figure 5 shows an example that highlights the top-10 words that LR-TFIDF focuses on. Negative feature importance scores indicate that the words do not support the prediction, while positive feature importance scores indicate that the words support the predicted topics. In this case, we find that most of the words negatively contribute to the top-1 predicted topic "quanx". As a consequence, the predicted probability for "quanx" is just 0.48.

Unique topic predictions. Besides analyzing the explanations of predictions, we also investigate the number of unique topics in top-10 predictions by LR-TFIDF and ZestXML for the whole test dataset. Out of 13,454 unique topics, we found that there are only 3,057 unique topics appearing in the top-10 predictions produced by LR-TFIDF. Meanwhile, the corresponding number for ZestXML is larger with 11,801 ($\sim$88%) unique topics appearing in top-10 predictions, nearly 4 times that of LR-TFIDF. This suggests that LR-TFIDF is biased toward a small number of topics while ZestXML is

[6]https://github.com/davebiagioni/pylighter



(a) Feature importance scores from LR-TFIDF. Its top-1 prediction is "cpp"

(b) Feature importance scores from ZestXML. Its top-1 prediction is "hobbes"

Fig. 4: Explanation for LR-TFIDF's and ZestXML's predictions on a repository with the correct topics: "hobbes", "haskell-application"

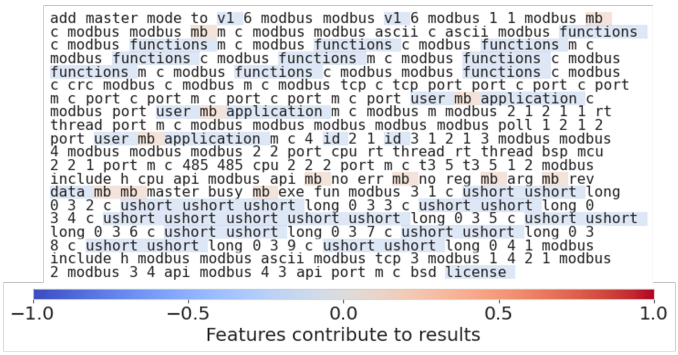


Fig. 5: Explanation for LR-TFIDF's prediction on a repository with the correct topics: "modbus", "master", "freemodbus"

able to predict a larger number of topics.

Zero-shot labels. ZestXML also shows greater capability in identifying topics that did not appear in the training dataset, compared with LR-TFIDF. Particularly, out of 5,188 unseen topics, 2,514 topics appear in the top-10 predictions of ZestXML, while LR-TFIDF does not predict any. More surprisingly, ZestXML successfully predicts 1,593 of these unique unseen topics out of 2,514, which shows the promising use of ZestXML on the zero-shot setting.

VII. Threats to Validity

Threats to internal validity. A source of threat to internal validity is the possibility of error in our implementation. We have checked our implementation to ensure its correctness, and we have provided the detailed parameters for the training of the XML models in Section IV. We also open-source our implementation [42]. By using the parameters and code that are available, other researchers can validate our findings. Thus, we believe that the threat is minimal.

Threats to construct validity. A source of threat to construct validity is the suitability of the metrics that we use for our evaluation. To minimize threats from this potential issue, we used metrics that are standard and frequently used for the multi-label classification experiments [6], [11], [12], [37] including precision, recall, F1, and success rate.

Threats to external validity. One threat is related to the splitting of the dataset. One approach to address this is to run multiple experiments with different data splits. Unfortunately, LightXML and LR-TFIDF require a longer time for training (up to 48 hours) as the number of unique topics is high (i.e., 37,161 unique topics). Therefore, a comparison of the models using multiple split data is expensive. Thus, in our experiments, we follow the experiment design from previous studies that have done an evaluation on XML methods [24], [27], [43] and work on topic recommendation [6].

VIII. Conclusion and Future Work

Automatically recommending topics for software repositories can improve their discoverability. Due to a large number of topics and the long tail distribution of topics, the identification of relevant topics for each repository is a challenge. This challenge was overlooked by prior studies on this subject. In this study, we collect a new dataset consisting of 21,273 repositories with 37,161 unique topics. Different from prior studies, we do not filter out topics that occur with low frequency. This results in a dataset with large number of infrequently occurring topics. As the task shares characteristics of Extreme Multi-Label Learning (XML) problems, we formulate the problem as an XML problem and experiment with several models. The best results come from a zero-shot XML approach (i.e., ZestXML), which improves the average F1 score of a previously proposed approach by 17.35%.

For future work, we plan to utilize more data that is available, such as the source code and code documentation from the repositories. We also plan to investigate the effectiveness of XML and zero-shot learning to other tasks, e.g., bug management [44], [45]. Moreover, we plan to perform an empirical study on the tagging culture in GitHub, c.f., [46].

Replication package: Our dataset and implementation are available at https://figshare.com/s/dc6d69629442c6ac3bbb.

REFERENCES

[1] F. Thung, T. F. Bissyande, D. Lo, and L. Jiang, "Network structure of social coding in github," in *2013 17th European conference on software maintenance and reengineering*. IEEE, 2013, pp. 323–326.

[2] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *Proceedings of the 36th International Conference on Software engineering (ICSE)*, 2014, pp. 356–366.

[3] G. Pinto, I. Steinmacher, and M. A. Gerosa, "More common than you think: An in-depth study of casual contributors," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 112–123.

[4] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 conference on computer supported cooperative work*, 2012, pp. 1277–1286.

[5] H. S. Qiu, A. Nolte, A. Brown, A. Serebrenik, and B. Vasilescu, "Going farther together: The impact of social capital on sustained participation in open source," in *2019 ieee/acm 41st International Conference on Eoftware Engineering (ICSE)*. IEEE, 2019, pp. 688–699.

[6] M. Izadi, A. Heydarnoori, and G. Gousios, "Topic recommendation for software repositories using multi-label classification algorithms," *Empirical Software Engineering*, vol. 26, no. 5, pp. 1–33, 2021.

[7] C. Treude and M.-A. Storey, "How tagging helps bridge the gap between social and technical aspects in software development," in *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 2009, pp. 12–22.

[8] ——, "Work item tagging: Communicating concerns in collaborative software development," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 19–34, 2010.

[9] X. Xia, D. Lo, X. Wang, and B. Zhou, "Tag recommendation in software information sites," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 287–296.

[10] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "Entagrec++: An enhanced tag recommendation system for software information sites," *Empirical Software Engineering*, vol. 23, no. 2, pp. 800–832, 2018.

[11] C. Di Sipio, R. Rubei, D. Di Ruscio, and P. T. Nguyen, "A multinomial naïve bayesian (mnb) network to automatically recommend topics for github repositories," in *Proceedings of the Evaluation and Assessment in Software Engineering*, 2020, pp. 71–80.

[12] J. Di Rocco, D. Di Ruscio, C. Di Sipio, P. T. Nguyen, and R. Rubei, "Hybridrec: A recommender system for tagging github repositories," *Applied Intelligence*, pp. 1–23, 2022.

[13] Q. Li, J. Song, D. Tan, H. Wang, and J. Liu, "Pdgraph: A large-scale empirical study on project dependency of security vulnerabilities," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 161–173.

[14] I. E. H. Yen, X. Huang, K. Zhong, P. Ravikumar, and I. S. Dhillon, "Pd-sparse: A primal and dual sparse approach to extreme multiclass and multilabel classification," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, p. 3069–3077.

[15] I. E. Yen, X. Huang, W. Dai, P. Ravikumar, I. Dhillon, and E. Xing, "Ppdsparse: A parallel primal-dual sparse method for extreme classification," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2017, p. 545–553.

[16] S. Khandagale, H. Xiao, and R. Babbar, "Bonsai: diverse and shallow trees for extreme multi-label classification," *Machine Learning*, vol. 109, 11 2020.

[17] Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma, "Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising," in *WWW '18: Proceedings of the 2018 World Wide Web Conference*, 04 2018, pp. 993–1002.

[18] Y. Prabhu and M. Varma, "FastXML: a fast, accurate and stable tree-classifier for extreme multi-label learning," *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014.

[19] M. Wydmuch, K. Jasinska, M. Kuznetsov, R. Busa-Fekete, and K. Dembczyński, "A no-regret generalization of hierarchical softmax to extreme multi-label classification," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 6358–6368.

[20] Y. Tagami, "Annexml: Approximate nearest neighbor search for extreme multi-label classification," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 455–464.

[21] K. Bhatia, H. Jain, P. Kar, M. Varma, and P. Jain, "Sparse local embeddings for extreme multi-label classification," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, p. 730–738.

[22] T. Jiang, D. Wang, L. Sun, H. Yang, Z. Zhao, and F. Zhuang, "LightXML: Transformer with dynamic negative sampling for high-performance extreme multi-label text classification," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 7987–7994, May 2021.

[23] J. Liu, W.-C. Chang, Y. Wu, and Y. Yang, "Deep learning for extreme multi-label text classification," in *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 115–124.

[24] N. Gupta, S. Bohra, Y. Prabhu, S. Purohit, and M. Varma, "Generalized zero-shot extreme multi-label learning," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 527–535.

[25] J. Di Rocco, D. Di Ruscio, C. Di Sipio, P. Nguyen, and R. Rubei, "Top-filter: an approach to recommend relevant github topics," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–11.

[26] F. Herrera, F. Charte, A. J. Rivera, and M. J. d. Jesus, "Multilabel classification," in *Multilabel Classification*. Springer, 2016, pp. 17–31.

[27] S. A. Haryono, H. J. Kang, A. Sharma, A. Sharma, A. Santosa, A. M. Yi, and D. Lo, "Automated identification of libraries from vulnerability data: Can we do better?" in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension (ICPC)*, 2022.

[28] Y. Lyu, T. Le-Cong, H. J. Kang, R. Widyasari, Z. Zhao, X.-B. D. Le, M. Li, and D. Lo, "Chronos: Time-aware zero-shot identification of libraries from vulnerability reports," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023. [Online]. Available: https://arxiv.org/abs/2301.03944

[29] Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma, "Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 993–1002.

[30] S. Khandagale, H. Xiao, and R. Babbar, "Bonsai: diverse and shallow trees for extreme multi-label classification," *Machine Learning*, vol. 109, no. 11, pp. 2099–2119, 2020.

[31] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *NAACL*, 2019.

[32] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *NeurIPS*, 2019.

[33] L. Zhuang, L. Wayne, S. Ya, and Z. Jun, "A robustly optimized bert pre-training approach with post-training," in *Proceedings of the 20th Chinese National Conference on Computational Linguistics*. Chinese Information Processing Society of China, 2021, pp. 1218–1227.

[34] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first Instructional Conference on Machine Learning*, vol. 242, no. 1. Citeseer, 2003, pp. 29–48.

[35] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo, "Categorizing the content of github readme files," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1296–1327, 2019.

[36] B. Xu, T. Hoang, A. Sharma, C. Yang, X. Xia, and D. Lo, "Post2vec: Learning distributed representations of stack overflow posts," *IEEE Transactions on Software Engineering*, 2021.

[37] X.-Y. Wang, X. Xia, and D. Lo, "Tagcombine: Recommending tags to contents in software information sites," *Journal of Computer Science and Technology*, vol. 30, no. 5, pp. 1017–1035, 2015.

[38] M. Izadi, A. Heydarnoori, and G. Gousios, "Software Tag Recommender," 2022. [Online]. Available: https://github.com/MalihehIzadi/SoftwareTagRecommender

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[40] N. Rethmeier and I. Augenstein, "Long-tail zero and few-shot learning via contrastive pretraining on and for small data," in *Computer Sciences & Mathematics Forum*, vol. 3, no. 1. MDPI, 2022, p. 10.

[41] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1135–1144.

[42] R. Widyasari, Z. Zhao, T. Le-Cong, H. J. Kang, and D. Lo, "GitHub Repository Topic Recommendation," 2022. [Online]. Available: https://figshare.com/s/dc6d69629442c6ac3bbb

[43] Y. Chen, A. E. Santosa, A. Sharma, and D. Lo, "Automated identification of libraries from vulnerability data," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, 2020, pp. 90–99.

[44] C. Sun, D. Lo, S. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, November 6-10, 2011*, P. Alexander, C. S. Pasareanu, and J. G. Hosking, Eds. IEEE Computer Society, 2011, pp. 253–262.

[45] S. Wang and D. Lo, "Version history, similar report, and structure: putting them together for improved bug localization," in *22nd International Conference on Program Comprehension, ICPC 2014, Hyderabad, India, June 2-3, 2014*, C. K. Roy, A. Begel, and L. Moonen, Eds. ACM, 2014, pp. 53–63.

[46] P. S. Kochhar, F. Thung, N. Nagappan, T. Zimmermann, and D. Lo, "Understanding the test automation culture of app developers," in *8th IEEE International Conference on Software Testing, Verification and Validation, ICST 2015, Graz, Austria, April 13-17, 2015*. IEEE Computer Society, 2015, pp. 1–10.