6-2017

Cataloging GitHub repositories

Abhishek SHARMA
Singapore Management University, abhisheksh.2014@phdis.smu.edu.sg

Ferdian THUNG
Singapore Management University, ferdiant.2013@phdis.smu.edu.sg

Pavneet Singh KOCHHAR
Singapore Management University, kochharps.2012@phdis.smu.edu.sg

Agus SULISTYA
Singapore Management University, aguss.2014@phdis.smu.edu.sg

David LO
Singapore Management University, davidlo@smu.edu.sg

Cataloging GitHub Repositories

Abhishek Sharma*
School of Information Systems
Singapore Management University,
Singapore
abhisheksh.2014@smu.edu.sg

Ferdian Thung*
School of Information Systems
Singapore Management University,
Singapore
ferdiant.2013@smu.edu.sg

Pavneet Singh Kochhar
School of Information Systems
Singapore Management University,
Singapore
kochharps.2012@smu.edu.sg

Agus Sulistya
School of Information Systems
Singapore Management University,
Singapore
aguss.2014@smu.edu.sg

David Lo
School of Information Systems
Singapore Management University,
Singapore
davidlo@smu.edu.sg

ABSTRACT

GitHub is one of the largest and most popular repository hosting service today, having about 14 million users and more than 54 million repositories as of March 2017. This makes it an excellent platform to find projects that developers are interested in exploring. GitHub showcases its most popular projects by cataloging them manually into categories such as DevOps tools, web application frameworks, and game engines. We propose that such cataloging should not be limited only to popular projects. We explore the possibility of developing such cataloging system by automatically extracting functionality descriptive text segments from readme files of GitHub repositories. These descriptions are then input to LDA-GA, a state-of-the-art topic modeling algorithm, to identify categories. Our preliminary experiments demonstrate that additional meaningful categories which complement existing GitHub categories can be inferred. Moreover, for inferred categories that match GitHub categories, our approach can identify additional projects belonging to them. Our experimental results establish a promising direction in realizing automatic cataloging system for GitHub.

KEYWORDS

GitHub, Latent Dirichlet Allocation, Genetic Algorithm

1 INTRODUCTION

GitHub is currently the largest repository hosting service, with more than 54 million repositories hosted there by March 2017[1].

A large number of open source projects, ranging from new to established ones, put their code on GitHub. GitHub provides an ecosystem for developers to work together to build software. Developers can contribute to projects of interest, and build more complex systems by reusing functionalities developed by others. To enable developers to contribute to projects and reuse functionalities well, they need to be aware of what is available on GitHub. Unfortunately, discovering projects related to topics one is interested in among the millions present in GitHub is not an easy task.

Recently, to aid information discovery, GitHub launched *showcases*[2], which presents hundreds of popular repositories grouped into categories. The *showcases* serves as a catalog that can help developers identify repositories of interest and navigate through GitHub. Unfortunately, it contains only a small minority of projects in GitHub. It was created manually and thus it is difficult to increase its size (either in the number of categories, or in the number of projects per category) substantially without much effort. Another feature introduced by GitHub to aid information discovery is *topics*[3] which allows GitHub users to add any word or tag which is related to their project. However in its present form this feature requires the tags to be added manually by project administrators. Also the administrators do not get any topic or tag recommendations of any kind and can also add any word as tag. In this work, to address the above challenges, we explore a topic modelling based semi-automatic approach for cataloging a large number of GitHub projects which can also be used to suggest topics for projects to project administrators.

Our approach works by processing *readme* files of GitHub projects. It first employs a heuristic to automatically extract descriptive text segments related to project functionality from the readme files. Irrelevant details from the readme files (e.g., installation procedure, copyright information, etc.) are dropped. Next, the text segments are input into a topic modeling algorithm. At the end of the previous step, we have a set of topics (i.e., categories) and documents (i.e., projects) that belong to them. Additionally, for each topic we have a set of words that characterize it. These words can be manually analyzed to identify a suitable name for the topic.

As the topic modeling algorithm, we use LDA-GA which was proposed by Panichella et al. and has been shown effective for

* The first 2 authors have contributed equally to the work.
[1] https://github.com/about

[2] https://github.com/showcases
[3] https://help.github.com/articles/classifying-your-repository-with-topics/

EASE'17, Karlskrona, Sweden

DOI: http://dx.doi.org/10.1145/3084226.3084287

a number of software engineering tasks [18]. LDA-GA combines two algorithms: Latent Dirichlet Allocation (LDA) and Genetic Algorithm (GA). LDA is a topic modeling algorithm which can be viewed as a document clustering solution. In our case, readme file of each GitHub project is a document, and we want to cluster these projects into categories. LDA however requires some parameters which affect its effectiveness. Instead of manually experimenting with different values for parameters in LDA, LDA-GA tunes these parameters by using Genetic Algorithm (GA) with Silhouette coefficient as the evaluation function. One downside of LDA-GA, is the need to run LDA many times which may take much time. To speed up the computation time of LDA-GA, instead of using standard batch LDA, we use an online LDA [7] that performs online learning for generating LDA model. It was shown that online LDA learns a topic model faster and the model is as good or better than the one produced by traditional batch LDA.

In this preliminary study, we experiment with 10,000 fairly popular projects on GitHub as input to our approach to investigate its potential to enhance the existing GitHub categories. We investigate the following research questions:

RQ1 How accurate is our description text extraction method?
RQ2 Can our proposed approach identify new categories that complement existing GitHub categories?
RQ3 Can our proposed approach identify new projects to existing GitHub categories?

The contributions of our work are as follows

(1) We present a new research problem on automatically cataloging GitHub.
(2) We propose an approach to solve the problem. It uses a similarity based approach to extract functionality descriptive text segments from *readme* files, and adapts LDA-GA to improve its efficiency.
(3) We demonstrate the potential of our approach to infer additional categories and additional projects to existing GitHub categories by an experiment on 10,000 GitHub projects.

The remainder of this paper is structured as follows. In Section 2, we describe our proposed approach in more details. In Section 3, we describe our experiment results. We list and describe related work in Section 4. Finally, we conclude and mention future work in Section 5.

2 APPROACH

The overall framework of our approach is illustrated in Figure 1. It takes as input a set of GitHub projects and output a set of categories in three main steps: data extraction, topic modelling, and manual analysis. The data extraction step recovers from the repositories texts that describe functionalities realized by code stored in them. The topic modeling step processes these functionality descriptive texts to generate a set of topics using LDA-GA proposed by Panichella et al. [18]. Each topic generated by LDA-GA corresponds to a ranked list of words characterizing the topics. LDA-GA outputs also inform us which projects are associated to each of the topics. The manual analysis step converts these topics (i.e., ranked lists of words) into categories. We describe these three steps in the following subsections.

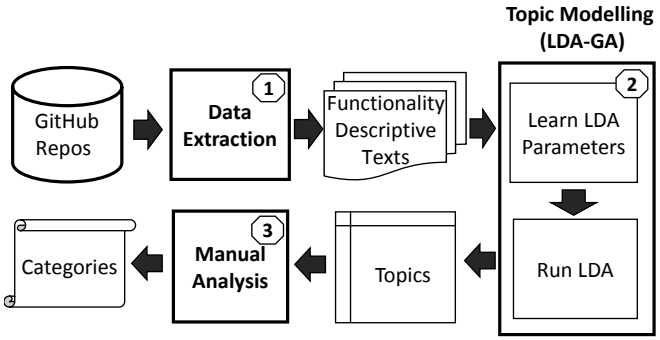

Figure 1: Overall Framework

2.1 Data Extraction

In this step, we use a combination of popular text preprocessing techniques and some heuristics to extract functionality descriptive texts GitHub repositories. We extract such texts from *readme* files of the repositories.

We first split a readme file into text segments. We observe many readme files contain headers and text in-between 2 headers often forms a coherent text segment. Based on this observation, we split each readme file into text segments, by considering text in between 2 successive headers as one segment. These text segments are then preprocessed and filtered.

After we have the text segments, we perform standard text preprocessing steps of *tokenization, stop word removal and stemming* on each of the segments:

(1) *Tokenization:* A text segment is broken into tokens where each token corresponds to a word that appears in the segment.
(2) *Stop Word Removal:* The common English stop words, such as "is", "are", etc, appear very often and thus may interfere with the topic generation when LDA is run. Thus, we remove these words using a list of English stop words provided at http://www.ranks.nl/stopwords.
(3) *Stemming:* After removing the stop-words, we reduce a word to its root form (e.g., "writing" and "written" are both reduced to "write") using a popular stemming algorithm, i.e., Porter stemmer [19]

Next, we use a heuristic to identify a segment which describes the *functionality* realized by the code in the repository. We want to filter out segments describing project installation, licensing, etc. To do this, we make use of the 1-2 line short description that is included in the homepage of a GitHub project – see Figure 2. These 1-2 line descriptions are too short to be processed by LDA (c.f., [8]), however, they can be used to identify longer text segments in the readme files. In particular, we measure the similarity between the short description and each text segment using Vector Space Model (VSM), and use this similarity to identify a functionality descriptive text segment.

Using VSM, each document (in our case the short description and a text segment being evaluated) is represented as a vector of weights. Each weight in the vector corresponds to a word in a document and the value of weight is calculated using the standard term frequency - inverse document frequency (TF-IDF) weighting scheme [15]. Next, we measure similarity between the description

Figure 2: A Sample Project Homepage on GitHub

and the text segment by computing the cosine similarity [15] of their corresponding weight vectors. For each readme file, we output the text segment that is the most similar to the short description as the descriptive text segment for the corresponding project.

2.2 Topic Modeling

For the topic modeling step, we use LDA-GA [18]. LDA-GA tunes LDA parameters using GA; these parameters include: (1) *numtopic*; (2) *numiter*; (3) α; and (4) β. For getting these parameters, we search *numtopic* in the range of [2, 50], *numiter* in the range of [20, 500], and both α and β in the range of (0, 0.1].

LDA-GA needs to run LDA many times as it searches for a good parameter setting. This is a time consuming process. To improve its efficiency, instead of using standard batch LDA, we use Online LDA [7], which learns topic model using online learning. Hoffman et al. have shown that online LDA speeds up the model learning significantly while still achieving a result as good as or better than batch LDA [7].

The topic modeling process proceeds as follows:

- First, given a project collection of n projects $\{p_1, ..., p_n\}$, in which each project is represented by its functionality descriptive text segment, we group the projects into their respective clusters. The cluster membership is determined by the project topic probability given by the LDA model that is generated using the current parameters given by GA. A project is assigned to the cluster that represents the most probable topic of that project. Thus, there are *numtopic* clusters.
- Second, for each project p_i, we calculate the distances between p_i with other projects within the same cluster. We then take the maximum of those distances and denote it as $a(p_i)$. We also calculate the distances between p_i and the centroids of other clusters (i.e., the clusters where p_i is not a member). The centroid of a cluster is an average of all the projects in the cluster. We take the minimum of those distances and denote it as $b(p_i)$.
- Third, given $a(p_i)$ and $b(p_i)$, we calculate the Silhouette coefficient $S(p_i)$ for project p_i as the fitness function for GA, by following the formula below.

$$S(p_i) = (b(p_i) - a(p_i)) \div (max(a(p_i), b(p_i)))$$

- Lastly, given the Silhouette coefficients $S(p_i)$ for all projects in $\{p_1, ..., p_n\}$, we calculate their mean and use this value as the fitness score for GA. The range of Silhouette coefficient is [-1, 1]. Higher value of the coefficient indicates that a better clustering is achieved.

We use *gensim*[4] implementation of Online LDA and *pyevolve*[5] implementation of GA. For GA, we set population size and generation to 100. For other parameters, we use default parameters given by both tools.

2.3 Manual Analysis

The output of the previous step is a set of topics along with a set of projects that belong to each topic. Each topic is represented by a ranked list of words that best represent it. Unfortunately, LDA does not output a good name for each topic. Moreover, the topics may not be ideal. Some topics may need to be merged, and some other topics do not correspond to a coherent concept and may need to be deleted.

This manual analysis step requires a data analyst to perform three actions:

- First, the data analyst needs to identify and delete incoherent topics. These topics may arise due to the imperfection of the data extraction or topic modeling step. The data extraction step may output a wrong text segment which describes other information instead of the functionality of the code implemented in a repository. GA may learn a semi-optimal number of topics. LDA may learn a poor topic.
- Next, the data analyst needs to infer a good representative name for each topic. The data analyst can get a hint of a good name for a topic by looking into the top words outputted by LDA and the projects that are associated with the topic.
- Finally, the data analyst needs to merge closely related topics into higher level topics. Topics inferred by LDA can be at different levels of granularity, and a data analyst may need to manually merge them.

3 PRELIMINARY EXPERIMENTS

3.1 Experimental Settings

In this preliminary experiments, we investigate the potential of our proposed approach using 10,000 fairly popular GitHub projects. In GitHub, developers can star a project and starring serves two purposes: it acts as a bookmark for the developers to keep a track of projects they are interested in, and also acts as a way to communicate appreciation of a project they like [1]. We used the GitHub search API[6] to identify the projects that have more than 20 stars, which returned 244,419 such projects. We then took a random sample of 10,000 projects. This is a statistically significant sample, considering a confidence level of 95% and a confidence interval of 1%. After we have identified the 10,000 projects, using GitHub API[7], we collected the metadata about these projects and also crawled their GitHub home pages.

In the manual analysis step, the authors discussed together what label would be best to explain the topic. The authors did not consider any particular dimensions such as language, platform, business

[4]https://radimrehurek.com/gensim/
[5]http://pyevolve.sourceforge.net/
[6]https://developer.github.com/v3/search/
[7]https://developer.github.com/v3//

domains, etc. They only considered top words produced by LDA-GA, along with projects that belong to a topic for assigning its label. LDA-GA returns 49 topics. A majority of the topics, 33 to be exact, can be labeled properly. We also merge some of these topics since they are coherent topics. We end up with a collection of 21 categories.

3.2 RQ1: How accurate is our description text extraction method?

To evaluate the effectiveness of our extraction approach, we randomly selected about 400 projects and extracted 2,570 text segments from their readme files. These text segments were then manually labeled as descriptive or non-descriptive by 7 pairs of labellers. The labellers are PhD students or software developers having a programming experience of more than 5 years. Each pair is assigned a set of text segments to label, and each person in a pair needs to initially label the text segments independently. The pairs agreed for segments (2,324 out of 2,570 segment). For the disagreement cases, each pair discusses them and decides the final ground truth label. The final distribution of the labels assigned is shown in Table 1.

Table 1: Agreements and Disagreements Among Labelers

Pair	Agreed	Disagreed
Pair-1	568	44
Pair-2	719	57
Pair-3	183	11
Pair-4	145	48
Pair-5	171	22
Pair-6	270	29
Pair-7	268	35
Total	2324	246

Based on the ground truth of labeled data created, we evaluated our approach to select descriptive text segments using the standard metrics of precision, recall, and F-measure. They are calculated based on four possible outcome: a text segment was correctly identified by our approach as descriptive (true positive, TP); a non-descriptive text segment was wrongly identified by our approach as descriptive (false positive, FP); a descriptive text segment was wrongly identified by our approach as non-descriptive (false negative, FN); and a non-descriptive text segment was correctly identified by our approach as descriptive (true negative, TN). The formulas used to compute precision, recall, and F-measure are:

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (3)$$

Tables 2 presents the confusion matrix, comparing the descriptive paragraphs identified by our approach vis-a-vis the ground truth created by manual labeling. Table 3 shows the precision, recall, and F-measure scores achieved by our extraction method.An F-measure of 0.7 or higher is often considered reasonably good in the literature [9, 11].

Table 2: Confusion Matrix

	Predicted Descriptive	Predicted Non-Descriptive
Actual Descriptive	271	112
Actual Non-Descriptive	114	2073
Total	385	2185

Table 3: Precision, Recall, and F-measure of Our Approach.

	Precision	Recall	F-measure
Performance	0.7075	0.7038	0.7057

Table 5: Sample projects belonging to Raspberry PI Category

No	URL
1	https://github.com/Asquera/raspberry-devbox
2	https://github.com/steve71/RasPiBrew
3	https://github.com/DamianEdwards/PiDnx
4	https://github.com/lanceseidman/PiCAST
5	https://github.com/PandoCloud/pando-embedded-framework

Table 6: Sample projects belonging to Monitoring Category

No	URL
1	https://github.com/scouter-project/scouter
2	https://github.com/tofumatt/FindTheProblems
3	https://github.com/laravel-notification-channels/pusher-push-notifications
4	https://github.com/kfdm/irssi-growl
5	https://github.com/fnando/notifier

3.3 RQ2: Can our proposed approach identify new categories that complement existing GitHub categories?

We list the categories that are identified in Table 4. The identified categories can be further divided into three kinds: new categories (marked by ^{N}), categories with minor overlap (marked by ^{M}), and categories with substantial overlap (marked with ^{S}). Six categories are new ones, five are those with minor overlap, while ten are substantially similar to categories appearing in GitHub *showcases*.

One of the new category is Raspberry PI, which is a credit-card sized single-board computer developed by Raspberry foundation. Many GitHub projects develop solutions running on Raspberry PI, which is getting more popularity and adoption. Table 5 highlights some of them. Another category is monitoring; this category captures projects that monitor the health of a software system and notify developers for issues. Some examples of projects belonging to these categories are shown in Table 6.

A number of categories minimally overlap with GitHub categories. For example, GitHub has *Fabric mobile developer tools* as a category. Projects in this category uses Fabric.io to build their mobile apps. Our category captures general Android apps not restricted to those developed using Fabric.io. As another example,

Table 4: Discovered Topics

Topic Name (*Top LDA Words*)	**Topic Name** (*Top LDA Words*)
^{N}**Raspberry PI** (*pi tool raspberri devic detect*)	^{N}**Monitoring** (*notif server monitor program*)
^{N}**Cloud Computing** (*amazon upload solut s3 aw*)	^{N}**Blogging** (*menu blog code item engin*)
^{N}**Servers and Networking** (*http server request proxi asynchron*)	^{N}**Middleware** (*rabbitmq servic oracl web project*)
^{M}**Android Development** (*app android io librari support*)	^{M}**Ruby Related** (*gem log rubi rail support*)
^{M}**React Framework** (*react icon redux action compon*)	^{M}**Lua** (*studio code lua visual support*)
^{M}**Testing Frameworks** (*test pull unit fork request*)	
^{S}**Music** (*music applic project play button*)	^{S}**Web Frontend** (*css control contain view will*)
^{S}**Data Management and Analysis** (*data dataset analysi*)	^{S}**Rendering and Viewers** (*output color build render imag font*)
^{S}**Build and Productivity Tools** (*vagrant vertx sbt maven*)	^{S}**Operating Systems** (*window os develop linux applic*)
^{S}**Gaming and Chat Engines** (*game materi bot irc chat*)	^{S}**Web Frameworks** (*twitter grid field bootstrap form*)
^{S}**Text Editors** (*text progress sublim view bar*)	^{S}**Security** (*user databas configur server*)

GitHub has a category *Programming Languages* and in it several programming languages developed in GitHub are listed (e.g., the main repository for Ruby[8]. Our categories include *Ruby* and *Lua* and they capture not only the main repositories developing these languages, but also additional repositories.

A number of categories substantially match those specified by GitHub *showcases*. For example, GitHub has repositories *Music*, *Web Application Frameworks*, and *Open Source Operating Systems* which substantially match *Music*, *Web Frameworks*, and *Operating Systems* which appear in Table 4.

3.4 RQ3: Can our approach identify additional projects to existing categories?

We check if our approach can identify additional projects belonging to the above-mentioned five categories, and find that we indeed can find additional projects. The list of additional projects for each of the five categories are shown in Table 7.

3.5 Threats to Validity

Threats to internal validity are related to potential errors and biases that may have occurred when performing the experiments. We tried to minimize these threats by using third party tools for doing LDA and GA rather than implementing them ourselves. Usage of well-known third-party tools minimizes the probability of error. We have also checked our own code for cluster membership and calculating fitness score. We need to admit that the manual analysis step is subjective. We have tried to reduce this subjectivity by involving three people in the process. Threats to external validity are related to the generalizability of our result. We have performed a preliminary experiment with 10,000 GitHub projects, which is a statistically significant sample for GitHub projects with 20 stars or more. We plan to reduce this threat further by investigating more projects in the future.

Table 7: Sample Additional Projects Discovered

Category: Music
https://github.com/SmartThingsUle/DLNA-PLAYER
https://github.com/benkaiser/stretto
https://github.com/revolunet/VLCcontrols
https://github.com/eddturtle/TurtlePlayer
https://github.com/yoichitgy/EECircularMusicPlayerControl
Category: Web Frameworks
https://github.com/nephila/djangocms-blog
https://github.com/chaijs/chai
https://github.com/grommet/grommet
https://github.com/erik5388/jquery.gracket.js
https://github.com/Cmdv/React-RxJS
Category: Operating Systems
https://github.com/SharpCoder/rpi-kernel
https://github.com/pguyot/Einstein
https://github.com/krinkinmu/aufs
https://github.com/tornewuff/pycorn
https://github.com/mbaltaks/doublecommand
Category: Text Editors
https://github.com/jclement/SublimePandoc
https://github.com/dhategan/brackets-grunt
https://github.com/mgussekloo/Tabright
https://github.com/kimpettersen/random-sublime-text-plugin
https://github.com/chuyik/brackets-snippets
Category: Security
https://github.com/xyproto/permissions2
https://github.com/peredurabefrog/phpSecureLogin
https://github.com/benbahrenburg/Securely
https://github.com/robregonm/yii2-auth
https://github.com/ahoward/middleman-gibberish

4 RELATED WORK

LDA [3] has been used in a variety of SE domains for topic identification. Hindle et al. use LDA on commit log messages to understand what work has been done by developers and topics of development over time [6]. Neuhaus and Zimmermann use description texts from vulnerability reports in the Common Vulnerability and Exposures (CVE) database and apply LDA to find vulnerability types and new trends [17]. Thomas et al. use LDA to find out topics to understand evolution of a software system and perform a qualitative analysis on 12 releases of JHotDraw and compute several metrics [22]. Panichella et al. developed a genetic algorithm based approach to identify best settings for running LDA on software engineering domain [18].

Automatic categorization of software applications has also been explored in [14, 16] which assign projects into predefined categories based Application Programming Interface (API) calls made by the

[8]https://github.com/ruby/ruby

projects. In [10, 24] LDA has been used on the identifier names and comments in the project code for categorization. Our work, is different from above works as we do not assume some predefined categories and also use a high level and descriptive source of information (i.e., readme files). We also design a solution to deal with noise in readme files. Also instead of using normal LDA we used LDA-GA proposed by Panichella et al. (a combination Latent Dirichlet Allocation (LDA) and Genetic Algorithm (GA)) and evaluate it on a large number of projects. Our algorithm shows promise in finding new categories that can complement categories in GitHub *showcases* and can also be used to find additional projects belonging to the existing categories.

LDA has been also used to analyze and identify patterns in the way software developers use social channels. Barua et al. use LDA on a dataset of questions and answers from Stack Overflow to find topics among developer discussions and trends of these topics over time [2]. Vasques et al. had used LDA on Stack Overflow posts to get an insight into mobile development issues [13]. LDA was used to identify popular software engineering categories on Twitter [21]. Recently Yang et al. applied LDA to study what kind of security questions do developers ask in Stack Overflow [26].

GitHub has been one of the popular software repository being explored in software engineering research. Casalnuovo et al. analyze C and C++ projects from GitHub to understand the usage of asserts and their impact on defect occurrence in these projects [4]. Gousios et al. analyze 291 projects from GitHub which are written in various different programming languages to understand the pull-based software development model [5]. There also have been studies investigating relationship between programming languages and code quality based on GitHub projects [12, 20]. Thung et al. [23] had analyzed the network structure of GitHub developers to understand characteristics of influential projects and developers. Vasilescu et al. collect thousands of projects from GitHub to understand the rate and breadth of developers' context-switching behavior and the effect of context-switching on productivity [25]. Different that the above studies, the goal of our work is to catalog Github repositories by expanding GitHub *showcases*.

5 CONCLUSION AND FUTURE WORK

In this paper, we propose a technique to semi-automatically catalog GitHub projects. Our intention is to extend and complement GitHub *showcases*. We first develop an approach to find functionality descriptive text segments from *readme* files of GitHub projects. Next, we process these segments using LDA-GA to learn a set of topics, along with projects belonging to these topics. These topics are then manually analyzed to produce a set of categories. Our preliminary study on 10,000 GitHub projects demonstrate that: (1) Our approach can retrieve functionality descriptive texts from read me files with an F-measure of 0.7; (2) Our approach can identify new categories that are not captured by GitHub *showcases*; (3) Our approach can also identify new projects for existing GitHub categories.

For future work, we plan to experiment with a larger set of GitHub projects. Larger data can help us to uncover new topics that might not have been captured in our current analysis. We also plan to improve the accuracy of various steps of our proposed approach, and make it more automated. We also plan to evaluate our approach directly with software community on GitHub, by suggesting category labels to some projects and getting feedback on the same. We will also try to get ground truth topics for some projects which can then be used to measure the accuracy of our model using metrics such as precision and recall.

REFERENCES

[1] GitHub documentation. https://help.github.com/articles/about-stars/ [Online; accessed 19-Oct-2016].

[2] Anton Barua, Stephen W. Thomas, and Ahmed E. Hassan. 2014. What Are Developers Talking About? An Analysis of Topics and Trends in Stack Overflow. *Empirical Software Engineering* 19, 3 (2014), 619–654.

[3] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *the Journal of machine Learning research* 3 (2003), 993–1022.

[4] Casey Casalnuovo, Prem Devanbu, Abilio Oliveira, Vladimir Filkov, and Baishakhi Ray. 2015. Assert Use in GitHub Projects. In *ICSE*. 755–766.

[5] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An Exploratory Study of the Pull-based Software Development Model. In *ICSE*. 345–355.

[6] Abram Hindle, Michael W. Godfrey, and Richard C. Holt. 2009. What's hot and what's not: Windowed developer topic analysis. In *ICSM*. 339–348.

[7] Matthew Hoffman, Francis R Bach, and David M Blei. 2010. Online learning for latent dirichlet allocation. In *Advances in Neural Information Processing Systems*. 856–864.

[8] Liangjie Hong and Brian D. Davison. 2010. Empirical Study of Topic Modeling in Twitter. In *SOMA*.

[9] Neil Ireson, Fabio Ciravegna, Mary Elaine Califf, Dayne Freitag, Nicholas Kushmerick, and Alberto Lavelli. 2005. Evaluating machine learning for information extraction. In *ICML*. ACM, 345–352.

[10] Shinji Kawaguchi, Pankaj K Garg, Makoto Matsushita, and Katsuro Inoue. 2006. Mudablue: An automatic categorization system for open source repositories. *Journal of Systems and Software* 79, 7 (2006), 939–953.

[11] Eric Knauss, Siv Houmb, Kurt Schneider, Shareeful Islam, and Jan Jürjens. 2011. Supporting requirements engineers in recognising security issues. In *Requirements Engineering: Foundation for Software Quality*. Springer, 4–18.

[12] P. S. Kochhar, D. Wijedasa, and D. Lo. 2016. A Large Scale Study of Multiple Programming Languages and Code Quality. In *SANER*, Vol. 1. 563–573.

[13] Mario Linares-Vásquez, Bogdan Dit, and Denys Poshyvanyk. 2013. An exploratory analysis of mobile development issues using stack overflow. In *WCRE*. IEEE Press, 93–96.

[14] Mario Linares-Vásquez, Collin McMillan, Denys Poshyvanyk, and Mark Grechanik. 2014. On using machine learning to automatically classify software applications into domain categories. *Empirical Software Engineering* 19, 3 (2014), 582–618.

[15] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

[16] Collin McMillan, Mario Linares-Vasquez, Denys Poshyvanyk, and Mark Grechanik. 2011. Categorizing software applications for maintenance. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*. IEEE, 343–352.

[17] Stephan Neuhaus and Thomas Zimmermann. 2010. Security Trend Analysis with CVE Topic Models. In *ISSRE*. 111–120.

[18] Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. 2013. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *ICSE*. IEEE Press, 522–531.

[19] Martin F Porter. 1980. An algorithm for suffix stripping. *Program* 14, 3 (1980), 130–137.

[20] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2014. A large scale study of programming languages and code quality in github. In *FSE*. ACM, 155–165.

[21] Abhishek Sharma, Yuan Tian, and David Lo. 2015. What's hot in software engineering Twitter space?. In *ICSME, 2015*. IEEE, 541–545.

[22] Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, and Dorothea Blostein. 2010. Validating the Use of Topic Models for Software Evolution. In *SCAM*. 55–64.

[23] Ferdian Thung, Tegawende F Bissyande, David Lo, and Lingxiao Jiang. 2013. Network structure of social coding in github. In *CSMR), 2013*. IEEE, 323–326.

[24] Kai Tian, Meghan Revelle, and Denys Poshyvanyk. 2009. Using latent dirichlet allocation for automatic categorization of software. In *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*. IEEE, 163–166.

[25] Bogdan Vasilescu, Kelly Blincoe, Qi Xuan, Casey Casalnuovo, Daniela Damian, Premkumar Devanbu, and Vladimir Filkov. The Sky is Not the Limit: Multitasking Across GitHub Projects.

[26] Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. 2016. What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts. *Journal of Computer Science and Technology* 31, 5 (2016), 910–924.