The Name of the Title is Hope

Juri Di Rocco
Universitá degli studi dell'Aquila
juri.dirocco@univaq.it

ABSTRACT

GitHub has become a precious service for storing and managing software source code. Over the last year, 10M new developers have joined the GitHub community, contributing to more than 44M repositories. In order to help developers increase the reachability of their repositories, in 2017 GitHub introduced the possibility to classify them by means of topics. However, assigning wrong topics to a given repository can compromise the possibility of helping other developers reach it and eventually contribute to its development. In this paper, we present <RECOMMENDER ACRONYM>, a recommender system to assist open source software developers in selecting suitable topics to the repositories. <RECOMMENDER ACRONYM> exploits a collaborative filtering technique to recommend libraries to developers by relying on the set of dependencies, which are currently included in the project being This paper show...

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

datasets, collaborative filtering, topic recommender

ACM Reference Format:
Juri Di Rocco. 2018. The Name of the Title is Hope. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In

2 RELATED WORK

rw

3 PROPOSED APPROACH

In this section we describe <RECOMMENDER ACRONYM>, a system for providing developers with GitHub topics related recommendations. More specifically, <RECOMMENDER ACRONYM> is a *recommender system* [3] that encodes the relationships among various OSS artifacts by means of a semantic graph and utilizes a collaborative filtering technique [26] to recommend GitHub topics. Such a technique has been originally developed for e-commerce systems to exploit the relationships among users and products to predict the missing ratings of prospective items [12]. The technique is based on the premise that *"if users agree about the quality or relevance of some items, then they will likely agree about other items"* [26]. Our approach exploits this premise to solve the problem of library recommendation [22]. Instead of recommending goods or services to customers, we recommend third-party libraries to projects using an analogous mechanism: *"if projects share some libraries in common, then they will probably share other common libraries."*

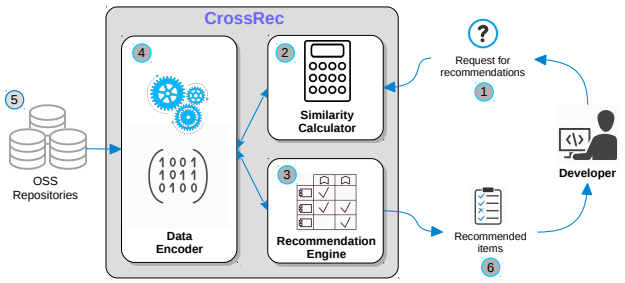

Figure 1: Overview of the <RECOMMENDER ACRONYM> Architecture.

To this end, the architecture of <RECOMMENDER ACRONYM> is shown in Fig. 1, and consists of the software components supporting the following activities:

- *Representing the relationships* among projects and libraries retrieved from existing repositories;
- *Computing similarities* to find projects, which are similar to that under development; and
- *Recommending libraries* to projects using a collaborative-filtering technique.

In a typical usage scenario of <RECOMMENDER ACRONYM>, we assume that a developer is creating a new system, in which she has already included some topics, or else is evolving an existing system. As shown in Fig. 1, the developer interacts with the system by sending a request for recommendations ①1. Such a request contains a list of libraries that are already included in the project the developer is working on. The `Data Encoder` ①4 collects *background data* from OSS repositories ①5, represents them in a graph, which is then used as a base for other components of <RECOMMENDER ACRONYM>. The `Similarity Calculator` module ①2 computes similarities among projects to find the most similar ones to the given project. The `Recommendation Engine` ①3 implements a *collaborative-filtering* technique [3],[29], it selects top-*k* similar projects, and performs computation to generate a ranked list of *top-N* libraries. Finally, the recommendations ①6 are sent back to the developer.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Woodstock '18, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

Background data has been collected GitHub [1]. The current version of <RECOMMENDER ACRONYM> [15] supports data extraction from GitHub, even though the support for additional platforms is under development.

In the following, the components `Data Encoder`, `Similarity Calculator`, and `Recommendation Engine` are singularly described.

3.1 Data Encoder

A recommender system for online services is based on three key components, namely *users*, *items*, and *ratings* [25],[19]. A *user-item ratings matrix* is built to represent the mutual relationships among the components. Specifically, in the matrix a user is represented by a row, an item is represented by a column and each cell in the matrix corresponds to a rating given by a user for an item [19]. For library recommendation, instead of users and items, there are projects and third-party libraries, and a project may include various libraries to implement desired functionalities. We derive an analogous user-item ratings matrix to represent the *project-library inclusion* relationships, denoted as $\ni$. In this matrix, each row represents a project and each column represents a library. A cell in the matrix is set to 1 if the library in the column is included in the project specified by the row, it is set to 0 otherwise. For the sake of clarity and conformance, we still denote this as a user-item ratings matrix throughout this paper.

For explanatory purposes, we consider a set of four projects $P = \{p_1, p_2, p_3, p_4\}$ together with a set of libraries $L = \{lib_1$=*junit:junit*; lib_2=*commons-io:commons-io*; lib_3=*log4j:log4j*; lib_4=*org.slf4j:slf4j-api*; lib_5=*org.slf4j:slf4j-log4j12*$\}$. By observing the *pom.xml* files of the projects in P, we discovered the following inclusions: $p_1 \ni lib_1, lib_2$; $p_2 \ni lib_1, lib_3$; $p_3 \ni lib_1, lib_3, lib_4, lib_5$; $p_4 \ni lib_1, lib_2, lib_4, lib_5$. Accordingly, the user-item ratings matrix built to model the occurrence of the libraries is depicted in Fig. 2.

$$
\begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{array}
\begin{array}{c}
\begin{array}{ccccc} lib_1 & lib_2 & lib_3 & lib_4 & lib_5 \end{array} \\
\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix}
\end{array}
$$

Figure 2: An example of a user-item ratings matrix to model the inclusion of third-party libraries in OSS projects.

3.2 Similarity Calculator

The `Recommendation Engine` of <RECOMMENDER ACRONYM> works by relying on an analogous user-item ratings matrix. To provide inputs for this module, the first task of <RECOMMENDER ACRONYM> is to perform similarity computation on its input data to find the most similar projects to a given project. In this respect, the ability to compute the similarities among projects has an effect on the recommendation outcomes. Nonetheless, computing similarities among software systems is considered to be a difficult task [13]. In addition, the diversity of artifacts in OSS repositories as well as their cross relationship makes the similarity computation become

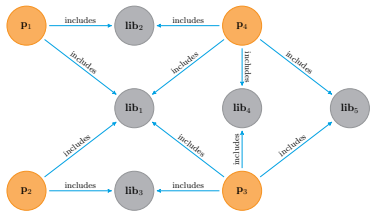

Figure 3: Graph representation for projects and libraries.

even more complicated. In OSS repositories, both humans (i.e., developers and users) and non-human actors (such as repositories, and libraries) have mutual dependency and implication on the others. The interactions among these components, such as developers commit to or star repositories, or projects include libraries, create a tie between them and should be included in similarity computation.

We assume that a representation model that addresses the semantic relationships among miscellaneous factors in the OSS community is beneficial to project similarity computation. To this end, we consider the community of developers together with OSS projects, libraries, and their mutual interactions as an *ecosystem*. We derive a *graph-based* model to represent different kinds of relationships in the OSS ecosystem, and eventually to calculate similarities. In the context of mining OSS repositories, the graph model is a convenient approach since it allows for flexible data integration and numerous computation techniques. By applying this representation, we are able to transform the set of projects and libraries shown in Fig. 2 into a directed graph as in Fig. 3. We adopted our proposed CrossSim approach [16],[17] to compute the similarities among OSS graph nodes. It relies on techniques successfully exploited by many studies to do the same task [8],[5]. Among other relationships, two nodes are deemed to be similar if they point to the same node with the same edge. By looking at the graph in Fig. 3, we can notice that p_3 and p_4 are highly similar since they both point to three nodes lib_1, lib_4, lib_5. This reflects what also suggested in a previous work by McMillan et al. [13], i.e., similar projects implement common pieces of functionality by using a shared set of libraries.

Using this metric, the similarity between two project nodes p and q in an OSS graph is computed by considering their feature sets [8]. Given that p has a set of neighbor nodes $(lib_1, lib_2, .., lib_l)$, the features of p are represented by a vector $\overrightarrow{\phi} = (\phi_1, \phi_2, .., \phi_l)$, with ϕ_i being the weight of node lib_i. It is computed as the *term-frequency inverse document frequency* value as follows:

$$\phi_i = f_{lib_i} \times log(\frac{|P|}{a_{lib_i}}) \tag{1}$$

where f_{lib_i} is the number of occurrence of lib_i with respect to p, it can be either 0 and 1 since there is a maximum of one lib_i connected to p by the edge *includes*; $|P|$ is the total number of considered projects; a_{lib_i} is the number of projects connecting to lib_i via the edge *includes*. Eventually, the similarity between p and q with their corresponding feature vectors $\overrightarrow{\phi} = \{\phi_i\}_{i=1,..,l}$ and $\overrightarrow{\omega} = \{\omega_j\}_{j=1,..,m}$ is computed as given below:

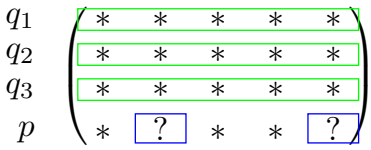

Figure 4: Computation of missing ratings using the user-based collaborative-filtering technique [29].

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

where n is the cardinality of the set of libraries that p and q share in common [8]. Intuitively, p and q are characterized by using vectors in an n-dimensional space, and Eq. 2 measures the cosine of the angle between the two vectors.

3.3 Recommendation Engine

Phuong ▶*Please reprhare this section*◀ The representation using a user-item ratings matrix allows for the computation of missing ratings [3],[19]. Depending on the availability of data, there are two main ways to compute the unknown ratings, namely *content-based* [21] and *collaborative-filtering* [14] recommendation techniques. The former exploits the relationships among items to predict the most similar items. The latter computes the ratings by taking into account the set of items rated by similar customers. There are two main types of collaborative-filtering recommendation: *user-based* [29] and *item-based* [25] techniques. As their names suggest, the user-based technique computes missing ratings by considering the ratings collected from similar users. Instead, the item-based technique performs the same task by using the similarities among items [6].

In the context of <RECOMMENDER ACRONYM>, the term *rating* is understood as the occurrence of a library in a project and computing missing ratings means to predict the inclusion of additional libraries. The project that needs prediction for library inclusion is called the *active project*. By the matrix in Fig. 4, p is the active project and an asterisk (∗) represents a known rating, either 0 or 1, whereas a question mark (?) represents an unknown rating and needs to be predicted.

Given the availability of the cross-relationships as well as the possibility to compute similarities among projects using the graph representation, we exploit the user-based collaborative-filtering technique as the engine for recommendation [12, 29]. Given an active project p, the inclusion of libraries in p can be deduced from projects that are similar to p. The process is summarized as follows:

- Compute the similarities between the active project and all projects in the collection;
- Select *top-k* most similar projects; and
- Predict ratings by means of those collected from the most similar projects.

The rectangles in Fig. 4 imply that the row-wise relationships between the active project p and the similar projects q_1, q_2, q_3 are exploited to compute the missing ratings for p. The following formula is used to predict if p should include l, i.e., $p \ni l$ [19]:

$$r_{p,l} = \overline{r_p} + \frac{\sum_{q \in topsim(p)}(r_{q,l} - \overline{r_q}) \cdot sim(p,q)}{\sum_{q \in topsim(p)} sim(p,q)} \quad (3)$$

where $\overline{r_p}$ and $\overline{r_q}$ are the mean of the ratings of p and q, respectively; q belongs to the set of *top-k* most similar projects to p, denoted as $topsim(p)$; $sim(p,q)$ is the similarity between the active project and a similar project q, and it is computed using Equation 2.

4 EVALUATION

This section describes the planning of our evaluation, having the *goal* of evaluating the performance of <RECOMMENDER ACRONYM>. In Section 4.1, we introduce the datasets exploited in our evaluation. The evaluation methodology and metrics are presented in Section **??**. Finally, Section 4.2 describes the research questions.

The evaluation process is depicted in Fig. 5 and it consists of three consecutive phases, i.e., *Data Preparation*, *Recommendation*, and *Outcome Evaluation*. We start with the *Data Preparation* phase by creating a dataset from GitHub projects. The dataset is then split into training and testing sets. In the *Recommendation* phase, given a project in the testing set, a portion of its libraries is extracted as ground-truth data, and the remaining is used as query to compute similarity and recommendations. Finally, by the *Outcome Evaluation* phase, we compare the recommendation results with those stored as ground-truth data to compute the quality metrics. All the aforementioned phases are detailed in the rest of this section.

4.1 Dataset Extraction

To evaluate the systems, we exploited a dataset with 6258 GitHub repositories that use 15757 topics.

By means of the GitHub API [2] we *randomly* collected a dataset consisting of 6,258 repositories. Claudio ▶*Add how the dataset has been mined (i.e., constraints in term of stars, forks, num of topic, etc)*◀ Juri ▶*Add Statistics*◀

To assess the performance of <RECOMMENDER ACRONYM> proposed approach, we applied ten-fold cross-validation, considering every time 9 folds (each one contains 625 projects) for training and the remaining one for testing. For every testing project p, a half of its libraries are *randomly* taken out and saved as ground truth data, let us call them *GT(p)*, which will be used to validate the recommendation outcomes. The other half are used as testing libraries or query, which are called ***te***, and serve as input for `Similarity Computation` and `Recommendation`. Though the ratio of query to ground-truth data can be arbitrarily set, we chose 50% for two reasons: *(i)* this value was used in the evaluation of the baselines in the original work [27],[20],[23]; and *(ii)* no matter how big a query is, the most important thing is to use exactly the same amount of data when evaluating the systems. In a separate experiment discussed later in this paper, the ratio of query to ground-truth has been varied to investigate whether an increase in the amount of testing data helps improve CrossRec's overall performance. The splitting mimics a real development process where a developer has already included some libraries in the current project, i.e., ***te*** and waits for recommendations, that means additional libraries to be incorporated. A recommender system is expected to provide her

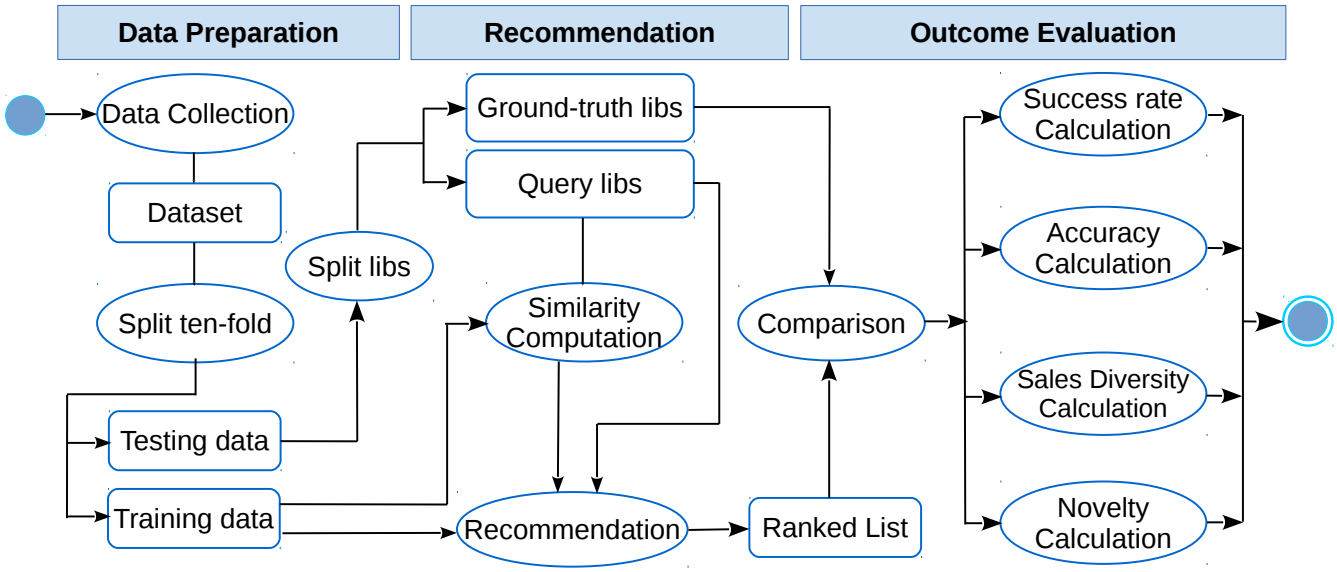

Figure 5: Evaluation Process.

with the other half, i.e., *GT(p)*. To ensure a reliable comparison between LibRec and <RECOMMENDER ACRONYM>, we performed cross-validation for both using exactly the same folds.

There are several metrics available to evaluate a ranked list of recommended items [19]. In the scope of this paper, *success rate*, *accuracy*, *sales diversity*, and *novelty* have been used to study the systems' performance as already proposed by Robillard et al. [22] and other studies [27],[18]. For a clear presentation of the metrics considered during the outcome evaluation, let us introduce the following notations:

- N is the cut-off value for the ranked list;
- k is the number of neighbor projects exploited for the recommendation process;
- For a testing project p, a half of its libraries are extracted and used as the ground-truth data named as $GT(p)$;
- $REC(p)$ is the *top-N* libraries recommended to p. It is a ranked list in descending order of real scores;
- If a recommended library $l \in REC(p)$ for a testing project p is found in the ground truth of p (i.e., $GT(p)$), hereafter we call this as a library *match* or *hit*.

If $REC_N(p)$ is the set of top-N items and $match_N(p) = GT(p) \bigcap REC_N(p)$ is the set of items in the *top-N* list that match with those in the ground-truth data, then the metrics are defined as follows.

Success rate@N. Given a set of testing projects P, this metric measures the rate at which a recommender system returns at least a library match among *top-N* items for every project $p \in P$ [27]:

$$success\ rate@N = \frac{count_{p \in P}(|match_N(p)| > 0)}{|P|} \tag{4}$$

where the function *count()* counts the number of times that the boolean expression specified in its parameter is *true*.

Accuracy. Accuracy is considered as one of the most preferred *quality indicators* for Information Retrieval applications [24]. However, *success rate@N* does not reflect how accurate the outcome of a recommender system is. For instance, given only one testing project, there is no difference between a system that returns 1 library match out of 5 and another system that returns all 5 library matches, since *success rate@5* is 100% for both cases (see Eq. (4)). Thus, given a list of *top-N* libraries, *precision@N* and *recall@N* are utilized to measure the *accuracy* of the recommendation results. *precision@N* is the ratio of the *top-N* recommended libraries belonging to the ground-truth dataset, whereas *recall@N* is the ratio of the ground-truth libraries appearing in the N recommended items [16],[8],[7]:

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

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

Sales Diversity. This term originates from the business domain where it is important to improve the coverage as also the distribution of products across customers, thereby increasing the chance that products will get sold by being introduced [28]. Similarly, in the context of library recommendation, *sales diversity* indicates the ability of the system to suggest to projects as much libraries as possible, as well as to disperse the concentration among all items, instead of focusing only on a specific set of libraries [22]. In the scope of this paper, *catalog coverage* and *entropy* are utilized to gauge *sales diversity*. Let L be the set of all libraries available for recommendation, $\#rec(l)$ denote the number of projects getting library l as a recommendation, i.e., $\#rec(l) = count_{p \in P}(|REC_N(p) \ni l|)$, $l \in L$, and $total$ denote the number of recommended items across all projects.

Catalog coverage measures the percentage of libraries being recommended to projects:

$$coverage@N = \frac{\left|\cup_{p \in P} REC_N(p)\right|}{|L|} \tag{7}$$

Entropy evaluates if the recommendations are concentrated on only a small set or spread across a wide range of libraries:

$$entropy = -\sum_{l \in L} \left(\frac{\#rec(l)}{total} \right) ln \left(\frac{\#rec(l)}{total} \right) \quad (8)$$

Novelty. In business, the *long tail effect* is the fact that a few of the most popular products are extremely popular, while the rest, so-called the long tail, is obscure to customers [4]. Recommending products in the long tail is beneficial not only to customers but also to business owners [28]. Given that the logarithmic scale is used instead of the decimal one on the x-Axis of Fig. ??, Fig. ??, and Fig. ??, the long tail effect can be spotted there as a few libraries are very popular by being included in several projects, whereas most of the libraries appear in fewer projects. Specifically, in D1 just 10 libraries are used by more than 200 projects, whereas 12, 962 libraries are used by no more than 10 projects. In D2 there are 3, 275 libraries, accounting for 63.85% of the total number, being used by no more than 10 projects. Similarly, in D3, 88.24% of the dependencies corresponding to 49, 799 libraries are used by no more than 10 projects. When recommending a library, *novelty* measures if a system is able to pluck libraries from the "long tail" to expose them to projects. This might help developers come across *serendipitous* libraries [9], e.g., those that are seen by chance but turn to be useful for their current project. To quantify *novelty*, we utilize *expected popularity complement* (EPC) which is defined in the following equation [28]:

$$EPC@N = \frac{\sum_{p \in P} \sum_{r=1}^{N} \frac{rel(p,r) * [1 - pop(REC_r(p))]}{log_2(r+1)}}{\sum_{p \in P} \sum_{r=1}^{N} \frac{rel(p,r)}{log_2(r+1)}} \quad (9)$$

where $rel(p,r) = |GT(p) \bigcap REC_r(p)|$ represents the relevance of the library at the r position of the *top-N* list to project p, $rel(p,r)$ can either be 0 or 1; $pop(REC_r(p))$ is the popularity of the library at the position r in the *top-N* recommended list. It is computed as the ratio between the number of projects that receive $REC_r(p)$ as a recommendation and the number of projects that receive the most ever recommended library as a recommendation. Equation 9 implies that the more unpopular libraries a system recommends, the higher the EPC value it obtains and vice versa.

When comparing results of LibRec and <RECOMMENDER ACRONYM>, we rely on suitable statistical procedures. Specifically, we use the Wilcoxon ranked sum test (considering one data point for each fold of the cross-validation) with a significance level of $\alpha = 5\%$, and the Cliff's delta (d) effect size measure [10]. Due to multiple tests being performed, we adjust p-values using the Holm's correction [11].

4.2 Research Questions

As mentioned in Section 4.1, a direct comparison between <RECOMMENDER ACRONYM> and LibFinder or LibCUP is difficult to perform, owing to the fact that there is no source code available for public use, as well as their re-implementation might not be compliant with the original work. Thus, we evaluate the recommendations by LibFinder, LibCUP and <RECOMMENDER ACRONYM> by experimenting <RECOMMENDER ACRONYM> on the corresponding datasets. Furthermore, we opt for *success rate* as it is the common metric used in the original work [27],[20],[23]. In contrast, the full access to its original implementation[1] allows us to painstakingly compare LibRec with <RECOMMENDER ACRONYM>, making use of all the metrics presented in this paper, i.e., *success rate*, *accuracy*, *sales diversity* and *novelty*.

In this sense, our empirical evaluation aims at addressing the following research questions:

- **RQ**$_1$: *How does <RECOMMENDER ACRONYM> compare with the state-of-the-art approaches in terms of success rate?* We evaluate <RECOMMENDER ACRONYM> by considering three state-of-the-art approaches, i.e., LibRec, LibFinder, and LibCUP, exploiting the datasets presented in Section 4.1. Furthermore, we study <RECOMMENDER ACRONYM> to see if it can recommend a specific version of a library, a requirement that all the baselines considered in this paper cannot satisfy.
- **RQ**$_2$: *How well can LibRec and <RECOMMENDER ACRONYM> recommend third-party libraries with respect to accuracy, sales diversity, and novelty?* Apart from *success rate*, we investigate if the approaches obtain a good performance in terms of *accuracy*, *sales diversity* and *novelty* [22],[18].
- **RQ**$_3$: *What are the reasons for the performance difference between LibRec and <RECOMMENDER ACRONYM>?* We are interested in understanding the rationale behind the performance differences between the two systems.
- **RQ**$_4$: *Does an increase in the amount of input data help improve CrossRec's overall performance?* We suppose that feeding <RECOMMENDER ACRONYM> with more data as query enhances the overall performance. Thus, rather than using $r = 50\%$ as the ratio of query to testing data, we varied r using incremental values, i.e., 20%, 40%, 60%, and 80% to analyze the change in performance.

We study the experimental results in the next section by referring to these research questions.

5 RESULTS

Result

6 CONCLUSIONS AND FUTURE WORK

conclusion

REFERENCES

[1] [n.d.]. GitHub. https://www.github.com. last accessed 16.06.2019.
[2] [n.d.]. GitHub REST API v3. https://developer.github.com/v3/. last accessed 16.06.2019.
[3] Charu Aggarwal. 2016. *Neighborhood-Based Collaborative Filtering*. Springer International Publishing, Cham, 29–70. https://doi.org/10.1007/978-3-319-29659-3_2
[4] Chris Anderson. 2006. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion.
[5] Cristian E. Briguez, Maximiliano C.D. Budán, Cristhian A.D. Deagustini, Ana G. Maguitman, Marcela Capobianco, and Guillermo R. Simari. 2014. Argument-based mixed recommenders and their application to movie suggestion. *Expert Systems with Applications* 41, 14 (2014), 6467 – 6482. https://doi.org/10.1016/j.eswa.2014.03.046
[6] Paolo Cremonesi, Roberto Turrin, Eugenio Lentini, and Matteo Matteucci. 2008. An Evaluation Methodology for Collaborative Recommender Systems. In *Proceedings of the 2008 International Conference on Automated Solutions for Cross Media*

[1]We gratefully acknowledge the LibRec source code implementation provided by Ferdian Thung and David Lo at the School of Information Systems, Singapore Management University

Content and Multi-channel Distribution (AXMEDIS '08). IEEE Computer Society, Washington, DC, USA, 224–231. https://doi.org/10.1109/AXMEDIS.2008.13

[7] Jesse Davis and Mark Goadrich. 2006. The Relationship Between Precision-Recall and ROC Curves. In *Proceedings of the 23rd International Conference on Machine Learning* (Pittsburgh, Pennsylvania, USA) *(ICML '06)*. ACM, New York, NY, USA, 233–240. https://doi.org/10.1145/1143844.1143874

[8] Tommaso Di Noia, Roberto Mirizzi, Vito Claudio Ostuni, Davide Romito, and Markus Zanker. 2012. Linked Open Data to Support Content-based Recommender Systems. In *Proceedings of the 8th International Conference on Semantic Systems* (Graz, Austria) *(I-SEMANTICS '12)*. ACM, New York, NY, USA, 1–8. https://doi.org/10.1145/2362499.2362501

[9] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. 2010. Beyond Accuracy: Evaluating Recommender Systems by Coverage and Serendipity. In *Proceedings of the Fourth ACM Conference on Recommender Systems* (Barcelona, Spain) *(RecSys '10)*. ACM, New York, NY, USA, 257–260. https://doi.org/10.1145/1864708.1864761

[10] Robert J. Grissom and John J. Kim. 2005. *Effect sizes for research: A broad practical approach* (2nd edition ed.). Lawrence Earlbaum Associates.

[11] S. Holm. 1979. A Simple Sequentially Rejective Bonferroni Test Procedure. *Scandinavian Journal on Statistics* 6 (1979), 65–70.

[12] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.Com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* 7, 1 (Jan. 2003), 76–80. https://doi.org/10.1109/MIC.2003.1167344

[13] Collin McMillan, Mark Grechanik, and Denys Poshyvanyk. 2012. Detecting Similar Software Applications. In *Proceedings of the 34th International Conference on Software Engineering* (Zurich, Switzerland) *(ICSE '12)*. IEEE Press, Piscataway, NJ, USA, 364–374. http://dl.acm.org/citation.cfm?id=2337223.2337267

[14] Catarina Miranda and Alípio M. Jorge. 2008. Incremental Collaborative Filtering for Binary Ratings. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01 (WI-IAT '08)*. IEEE Computer Society, Washington, DC, USA, 389–392. https://doi.org/10.1109/WIIAT.2008.263

[15] Phuong T. Nguyen, Juri Di Rocco, Davide Di Ruscio, and Massimiliano Di Penta. [n.d.]. CrossRec: Supporting Software Developers by Recommending Third-party Libraries - Online Appendix. https://github.com/crossminer/CrossRec. last accessed 25.09.2019.

[16] Phuong T. Nguyen, Juri Di Rocco, Davide Di Ruscio, Lina Ochoa, Thomas Degueule, and Massimiliano Di Penta. 2019. FOCUS: A Recommender System for Mining API Function Calls and Usage Patterns. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, Piscataway, NJ, USA, 1050–1060. https://doi.org/10.1109/ICSE.2019.00109

[17] P. T. Nguyen, J. Di Rocco, R. Rubei, and D. Di Ruscio. 2018. CrossSim: Exploiting Mutual Relationships to Detect Similar OSS Projects. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 388–395. https://doi.org/10.1109/SEAA.2018.00069

[18] Phuong T. Nguyen, Paolo Tomeo, Tommaso Di Noia, and Eugenio Di Sciascio. 2015. An Evaluation of SimRank and Personalized PageRank to Build a Recommender System for the Web of Data. In *Proceedings of the 24th International Conference on World Wide Web* (Florence, Italy) *(WWW '15 Companion)*. ACM, New York, NY, USA, 1477–1482. https://doi.org/10.1145/2740908.2742141

[19] Tommaso Di Noia and Vito Claudio Ostuni. 2015. Recommender Systems and Linked Open Data. In *Reasoning Web. Web Logic Rules - 11th International Summer School 2015, Berlin, Germany, July 31 - August 4, 2015, Tutorial Lectures*. 88–113. https://doi.org/10.1007/978-3-319-21768-0_4

[20] Ali Ouni, Raula Gaikovina Kula, Marouane Kessentini, Takashi Ishio, Daniel M. German, and Katsuro Inoue. 2017. Search-based Software Library Recommendation Using Multi-objective Optimization. *Inf. Softw. Technol.* 83, C (March 2017), 55–75. https://doi.org/10.1016/j.infsof.2016.11.007

[21] Michael J. Pazzani and Daniel Billsus. 2007. *Content-Based Recommendation Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 325–341. https://doi.org/10.1007/978-3-540-72079-9_10

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

[23] Mohamed Aymen Saied, Ali Ouni, Houari Sahraoui, Raula Gaikovina Kula, Katsuro Inoue, and David Lo. 2018. Improving reusability of software libraries through usage pattern mining. *Journal of Systems and Software* 145 (2018), 164 – 179. https://doi.org/10.1016/j.jss.2018.08.032

[24] Tefko Saracevic. 1995. Evaluation of Evaluation in Information Retrieval. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Seattle, Washington, USA) *(SIGIR '95)*. ACM, New York, NY, USA, 138–146. https://doi.org/10.1145/215206.215351

[25] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based Collaborative Filtering Recommendation Algorithms. In *Proceedings of the 10th International Conference on World Wide Web* (Hong Kong, Hong Kong) *(WWW '01)*. ACM, New York, NY, USA, 285–295. https://doi.org/10.1145/371920.372071

[26] J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. 2007. The Adaptive Web. Springer-Verlag, Berlin, Heidelberg, Chapter Collaborative Filtering Recommender Systems, 291–324. http://dl.acm.org/citation.cfm?id=1768197.1768208

[27] Ferdian Thung, David Lo, and Julia Lawall. 2013. Automated library recommendation. In *2013 20th Working Conference on Reverse Engineering (WCRE)*. 182–191. https://doi.org/10.1109/WCRE.2013.6671293

[28] Saúl Vargas and Pablo Castells. 2014. Improving sales diversity by recommending users to items. In *Eighth ACM Conference on Recommender Systems, RecSys '14, Foster City, Silicon Valley, CA, USA - October 06 - 10, 2014*. 145–152. https://doi.org/10.1145/2645710.2645744

[29] Zhi-Dan Zhao and Ming-sheng Shang. 2010. User-Based Collaborative-Filtering Recommendation Algorithms on Hadoop. In *Proceedings of the 2010 Third International Conference on Knowledge Discovery and Data Mining (WKDD '10)*. IEEE Computer Society, Washington, DC, USA, 478–481. https://doi.org/10.1109/WKDD.2010.54