Open Source Software Recommendations Using Github

Miika Koskela[1], Inka Simola[1], and Kostas Stefanidis[2]

[1] University of Tampere, Finland
{miika.s.koskela, inkariina.simola}@gmail.com
[2] University of Tampere, Finland
kostas.stefanidis@uta.fi

Abstract. The focus of this work is on providing an open source software recommendations using the Github API. Specifically, we propose a hybrid method that considers the programming languages, topics and README documents that appear in the users' repositories. To demonstrate our approach, we implement a proof of concept that provides recommendations.

1 Introduction

Recommender systems have become indispensable for several systems and Web sites, such as Amazon, Netflix, Yelp and Google News, helping users navigate through the abundance of available data items (e.g., [4,1,3,2,6]. In this paper[3], we introduce a recommender system for suggesting open source software using the Github API. Getting these recommendations requires users to have, in addition to a Github account, at least one of the following: (i) starred repositories, (ii) repositories followed, or (iii) users own repositories. As this information is public and available through Github's public API, recommendations can be generated to *any* user by *any* user. For generating recommendations, we exploit a hybrid method that combines three different similarity measures on three different feature sets. Specifically, we consider separately the languages found in a user's repositories, the topics present in the user's repositories, and the README documents of the user's repositories. To demonstrate our approach, we implement a proof of concept prototype for providing software recommendations.

2 Dataset

The dataset contains information on approximately 1000 software repositories and consists of languages, topics and README-files retrieved via Github's API. In order to recommend repositories some information on the users skills and interests is needed. We call this user information user profile. Specifically, the user profile consists of a combination of information from repositories the user

[3] The work was partially supported by the TEKES Finnish project Virpa D.

has somehow been associated with in the past. We consider the following as proof of an association between the user and a repository: (i) the user has starred the repository, (ii) the user has followed the repository, or (iii) the user has forked[4] the repository. For simplicity, we refer to these collectively as users repositories. For example, assume that user `mkoske` has starred the repository `php-ai/php-ml`, and has `mkoske/scatter-r` as his own repository. To construct a profile for this user, we collect the following pieces of information on these two repositories: (i) topics assigned to a repository by its owner, (ii) programming languages used in a repository, and (iii) README document of a repository. Next we will take a look at these pieces of information separately.

Topics: As an example, consider the `php-ai/php-ml`-repository, which is starred by user `mkoske`, and contains, among other topics, the following: php, machine-learning, classification, and data-science. Topics are specified manually by the repository owner or someone with proper permissions and is therefore a sparse source of information. In our sample dataset of 1000 items, almost half of the repositories (47,30%) are missing topics entirely.

Programming Languages: In the previous example, the `php-ai/php-ml`-repository contains two different languages: PHP and Shell. The latter seems to be used, e.g., to generate PHPUnit tests coverage reports and is therefore listed among the languages of the repository. The repository programming language is detected automatically[5] and no user interaction is required for that. In contrast, topics have to be input manually by the repository owner and are not detected automatically. In our sample dataset, over 95% of the repositories have a language or languages specified.

README: README-files are the third source of information on a repository. When accessing the repository at Github via web browser, the README is one of the first things that user encounters. It contains a free-form description of the repository in plain-text format and can be considered a front page for the repository. Table 1 shows some statistics on the lengths of the README files in our data set. The standard deviation is quite high, indicating that there is much variation in the lengths of the descriptions repository owners are assigning to their projects.

Statistic	Value
Min	115.00
Max	505 402.00
Mean	13 257.88
Median	6 891.00
Std	25 287.51

[4] https://help.github.com/articles/fork-a-repo/

[5] https://help.github.com/articles/about-repository-languages/

3 Method

We follow a hybrid method to produce open source software recommendations. Specifically, we combine three different similarity measures on three different feature sets to make a single list of recommendations. The final similarity score is a linear combination of three similarity scores: (i) all the languages found in a user's repositories are compared to the languages present in a given non-user repository, (ii) all the topics present in the user's repositories are compared to the topics present in a given non-user repository, and (iii) an averaged vector representation of the README documents of the user's repositories is compared to the vector representation of the README document of a given non-user repository.

We construct a language vector, a topic vector and a README vector for each repository. For the sake of practicality, all user repository vectors are collapsed together per type to form 3 different user vectors per user: a user language vector, a user topic vector and a user README-vector. Each of these vectors is then compared against the respective language, topic and README vector of each repository not associated with the user for whom a recommendation is to be generated. A linear combination of the resulting language, topic and README similarity scores is then calculated to obtain the final repository ranking.

Languages: The repository metadata contains information on the programming languages[6] used to write the software in question. Continuing with the example we had earlier, repository `php-ai/php-ml` is mainly written in PHP, but also contains some shell scripts. This is a small number of languages compared to, e.g., Visual Studio Code by Microsoft, which contains numerous languages.

In our dataset, languages detected in a repository were first transformed into binary vectors. If the repository contained a language, e.g., aforementioned PHP, the feature was assigned a value of 1, and 0 otherwise. The length of a repository language vector is the number of all programming languages found in the entire dataset, including the user repositories. Below is an example of a transformed binary language vector.

assembly	awk	c	…	typescript	vim_script	vb
1	0	1	…	0	1	0

The union of languages present in a user's repositories was used to form the user language vector. The mean of the user language vector was then subtracted from the user language vector, ensuring that cases where the user language vector lacks a language present in a repository language vector get a lower score than cases where neither user nor repository language vector contain a given language. This reflects the author's assumption that a user is most versed in languages present in his or her own repositories and not much else.

[6] Languages are automatically detected.

Ordinary cosine similarity does not differentiate between the aforementioned cases, and would give them equal language rankings. Adjusted cosine similarity, on the other hand, would needlessly penalize cases where the repository language vector lacks a language present in the user language vector. However, a repository does not have to contain all the user vector languages in order to be recommendable to the user. Hence, the mean was only subtracted from the user language vector and not the repository language vectors. For calculating the similarity between a user language vector and each repository language vector, we used a hybrid cosine similarity measure defined as: $sim_{cos}(a, b) = \frac{(a-\bar{a})\cdot b}{||(a-\bar{a})||\times||b||}$.

After subtracting the mean from the user language vector, our hybrid cosine similarity was calculated between the user language vector and each non-user repository language vector.

Topics: Repository topics resemble the commonly used *tags*: they contain at most a few words of free-form text that the author of the repository has chosen to describe the repository. Topics can also contain names of the programming languages used in writing the software. We did not filter out these potential overlaps and used the topics as they were. The length of a topic vector is the number of topics found in the entire dataset. Below is an example of a binary topic vector.

d	ad-blocker	admin	…	youtube	zeit	zsh
1	0	0	…	1	0	0

The union of topics present in a user's repositories was used to form the user topic vector. Jaccard similarity was then calculated between the user topic vector and each repository's topic vector. The Jaccard similarity measure, sometimes also called the *intersection over union* similarity, was used to compute similarities related to the topic component of the dataset. If a user vector contains (i.e., has values of 1 for) topics $a_0, a_1, \ldots, a_i$ and a repository vector contains topics $b_0, b_1, \ldots, b_j$, the Jaccard similarity between vectors $\mathbf{a}$ and $\mathbf{b}$ becomes: $sim_{jac}(\mathbf{a}, \mathbf{b}) = \frac{|\{a_0,a_1,\ldots a_i\}\cap\{b_0,b_1,\ldots b_j\}|}{|\{a_0,a_1,\ldots a_i\}\cup\{b_0,b_1,\ldots b_j\}|}$.

README: In our current dataset, all repositories have a README document. The READMEs were also retrieved using the Github API. Each repository README document was subjected to the following preprocessing operations: (i) tokenization, i.e., splitting the long string of text to tokens, (ii) removal of words with less than 3 characters, (iii) removal of content between any kind of brackets, (iv) removal of content matching certain frequently observed patterns (e.g., url, email address), (v) removal of English stopwords (e.g., 'and', 'when'), and (vi) part-of-speech tagging and removal of words that are not nouns in singular form.

A vector representation was then generated for each preprocessed README using TF-IDF as implemented in the TF-IDF-Vectorizer function of *Scikit-learn*:

– For each word appearing in the corpus, the documents containing the word are counted.

– Words appearing in just one or more than 95% of the documents of the corpus are removed.
– The inverse document frequency for each preserved word (~3000 words) is calculated.
– $IDF_i = \log \frac{\text{total \# documents}}{\text{\# of documents containing word i}}$.
– For each README, the normalized term frequency for each preserved word is calculated.
– The TF-IDF score (term_frequency x IDF score) is calculated for each preserved word in each README document, yielding a vectorized representation for each README document.

Below is an example of README-vector.

abilities	abort	…	zeros	zones	zookeeper
0.0	0.001	…	0.0	0.002	0.0

README vectors from the user's repositories were averaged to obtain a user README vector. Oridnary cosine similarity between the user README vector and each non-user-owned repository README vector was then calculated. It is possible for a repository README to not contain any of the words preserved by the TF-IDF transformation. In these cases, the cosine similarity between the user and repository README vectors is zero.

4 Recommendations

After feature extraction was completed, a linear combination of the language, topic and README similarities was calculated using weights w_l, w_t and w_r for language, topic and README respectively. For the time being and in the absence of user feedback, all weights were initialized to 0.33. The maximum and minimum scores thus assigned to the most and least recommended repositories respectively, were denoted *max_score* and *min_score*. The final scores were then obtained by normalizing the results using the following formula:

$$final\_score_{a,b} = \frac{w_l * lang\_sim_{a,b} + w_t * topic\_sim_{a,b} + w_r * readme\_sim_{a,b}}{max\_score - min\_score}.$$

Figure 1 shows the first five recommendations given to user `inkasimola` based on the public repositories on her Github account and 1000 retrieved repositories.

The project is public and located at: https://github.com/mkoske/recommender/. The software implementation was written in Python, while the *requests* and *Pandas* libraries were used for data retrieval from Github. As preprocessing - and especially part-of-speech tagging - entire README documents is time-consuming, it was done beforehand using Python scripts. Finally, the web application was written using *Flask*, and the Bootstrap CSS-framework.

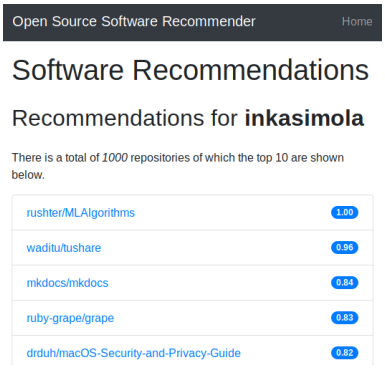

Fig. 1. Recommendations for user `inkasimola`.

5 Summary

In this paper, we propose a hybrid method that considers the programming languages, topics and README documents present in Github users' repositories, to generate open source software recommendations. The future work could evaluate the system by performing experiments with real users. Moreover, we could implement a model that takes into account the popularity of repositories and involves a feedback loop to allow for learning of user-specific feature weighting and further context-aware personalization of recommendations [7,5].

References

1. G. Adomavicius and Y. Kwon. Multi-criteria recommender systems. In *Recommender Systems Handbook*, pages 847–880. 2015.
2. M. Kyriakidi, K. Stefanidis, and Y. E. Ioannidis. On achieving diversity in recommender systems. In *ExploreDB*, 2017.
3. E. Ntoutsi, K. Stefanidis, K. Rausch, and H. Kriegel. Strength lies in differences: Diversifying friends for recommendations through subspace clustering. In *CIKM*, 2014.
4. J. J. Sandvig, B. Mobasher, and R. D. Burke. A survey of collaborative recommendation and the robustness of model-based algorithms. *IEEE Data Eng. Bull.*, 31(2):3–13, 2008.
5. K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Trans. Database Syst.*, 36(3):19:1–19:45, 2011.
6. K. Stefanidis and E. Ntoutsi. Cluster-based contextual recommendations. In *EDBT*, 2016.
7. K. Stefanidis, E. Pitoura, and P. Vassiliadis. Managing contextual preferences. *Inf. Syst.*, 36(8):1158–1180, 2011.