Finding Similar GitHub Reporsitories

by Jingci Wang

Abstract This project is to find similar GitHub reporsitories. Even though programming languages have certain criterions which make them understandable by machines, programmers have a rich usage of natural languages within their source code files through name entities or comments. In this project, I mainly dig similarity among GitHub repositories based on identifier names inside source code files.

Introduction

It would be beneficial to software developers to find relevant projects that can inspire them when performing their own works. It would also be easier for programmers to have existing functions at hand when building frameworks or structures. Maybe letting repositories themselves decide which ones are useful could achieve both efficiency and convenience.

Intuitively, finding relevant projects would give us some insights on how to recommend repositories to developers. Inspired by Markovtsev and Kant (2017), I decided to extract useful information from source code files in GitHub repositories and embed them as bag-of-words. Then, clustering exploration will be performed to find out how to decide the similarity among all those repositories. According to Markovtsev and Kant (2017), generally a source code file has the following components: keywords, identifiers, literals, comments and strings, and among them identifiers which contains some name entities should be the richest feature to discriminate the potential attributes of one project. Thus, my work would largely be dependent on the identifier names extracted from each source code files.

Data Preparation

First of all, I chose the top 2500 GitHub repositories with size less than 1000000 but great than 10000. To extract information from source code file, there are two available libraries to use. *Linguist* package is used for analyze the pragramming languages distribution within on repository. It can generate a list of source code file paths after cloning a given repository to the local device. Another one is *Pygments* which is used for syntax highlighter. *Pygments* can tokenize the code into lists of different components respectively based on the programming language of the given file. Both *Linguist* and *Pygments* can parse over 400 kinds of programming languages, but they only have around 200 in common. This will not be a problem since the overlapped set contains the ones that are most frequently used. I used the two packages to collect the identifiers and comments in source code files for each of the repository.

The next step is to clean the data. With the assistance of the algorithm in Markovtsev and Kant (2017), I split each extracted words according to some naming conventions. Then, stemming is performed while excluding the commonly English stop words. After that, I removed the repositories with too many words (larger than 1000000) and repositories with too few distinct words (less than 50), resulting in 1785 repositories. Now the idea is to treat each repository as one document, or more specifically, bag-of-words.

Evalution set

With the help of Lopes et al. (2017), I gained 956 pairs of repositories recording the number of files one cloned from another and the number of files affected in the repository being cloned. 150 different repositories appear in these 956 pairs. I simply determined the similarity between the two repositories to be the weighted percentage of cloned files in clone reporsitory and that of affected files in host reporitory weghting by the ratio of the number of files to the total number of files of the two repositories. Then, I used the AgglomerativeClustering methods to perform a bottom-up hierarchy clustering with respect to the similarities in each pair. The dendrogram is plotted as Figure 1

The horizontal axis represents different repos and the vertical axis the distance. From the graph we can see that if I set the cut-off to be around 0.5 (repositories with similarity larger than 0.5), the resulting would be few clusters with large number of members but rest clusters only contain few elements inside. Even though this cut-off setting can be subjective, two repositories with cloning percentage greater than 0.5 should be highly likely to be clustered into one group.

Topic Modeling

Topic modeling is a kind of statistical model for discovering the abstract "topics" that appear in a collection of documents, which can be seen as a kind of lantent factor. Topic modeling is very powerful

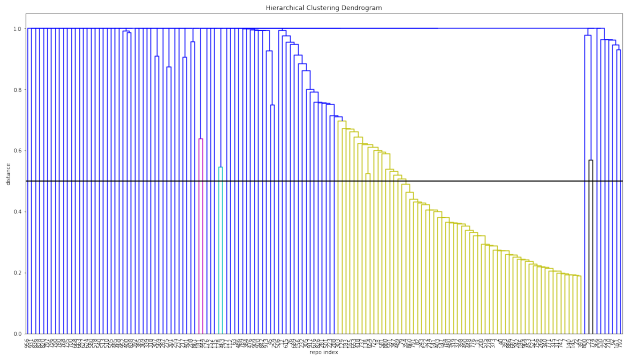

Figure 1: Dendrogram for Hierarchy Clustering

in terms of document clustering. Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA) are two typical examples of topic model. I used both in a small subset, but due to the computation limitation, I only perform LDA in the whole data I scraped. The general idea of LDA is to find the distribution of words over each latent topic and the distribution of topics over each document.

Thus, the first thing to do is to vectorize the collection of repositories. However, since the use of words in source code files are too diverse over each of the repositories, after applying CounterVectorizer, it gave over 150000 columns with sparsity larger than 99.6%. Thus, I only maintained the words appearring in more than 5 reporsitories but less than 75% of reporsitories, resulting in 45600 columns and 98% sparsity. Generally, using TF-IDF method to vectorize the collection of bag-of-words should outperform simply the CounterVectorizer. However, no matter how many latent topics I used, the outcome of all repositories only dominated by one same topic, which is apparently not practical. So I stayed with CounterVectorizer.

Considering both *Log-Likelihood* score and *Perplexity*, together with the independency of each topic, I obtained the probability distributions for each repositories over 30 topics. Then, I used applied *KMeans* clustering on the results, being aware that similar repositories should have similar topic distributions.

Applying grid search with respect to the number of groups to cluster and *Sum of Square Error* within each cluster as shown in Figure 2

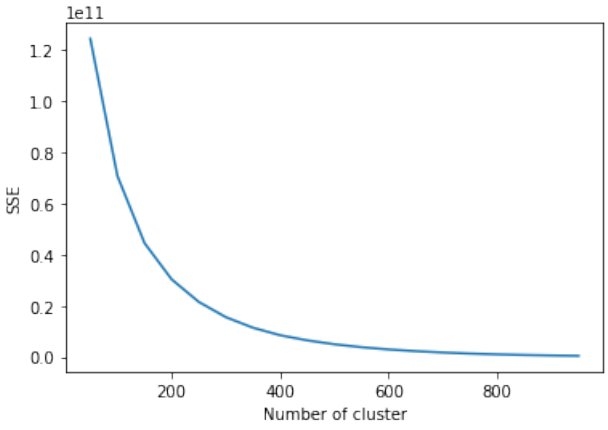

Figure 2: Grid Search for number of clusters with LDA

The plot indicates that 150 is a reasonable number of clusters for this dataset.

Hash Methods

The main problem of topic modeling method is that the embedding matrix after vectorization is far too sparse and giant. Looking at the data, I found that 75% of the bag-of-words contains less than 1400 different words. So I decided to choose a hash table with a small fixed length with each bucket recording the number of words hashed into it. The whole process is simply to use a prime number as a mod for hash value of each identifier. Then, each document can be transfered into a vector of same length.

I tested the method on a small number of manually labelled repositories. As shown in Figure 3, it turns out fairly good.

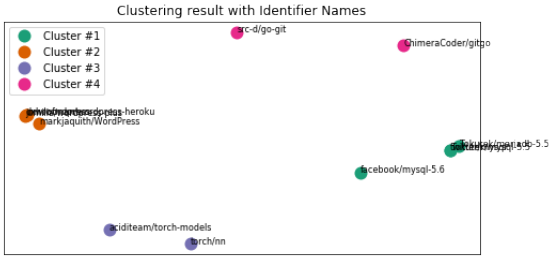

Figure 3: Clustering with Hash Method on a subset of labelled repos

The choices of length of hash tables for the whole set of repositories are illustrated in Figure 4. By randomly generating several prime numbers, the right-hand side is the grid search after performing normalization.

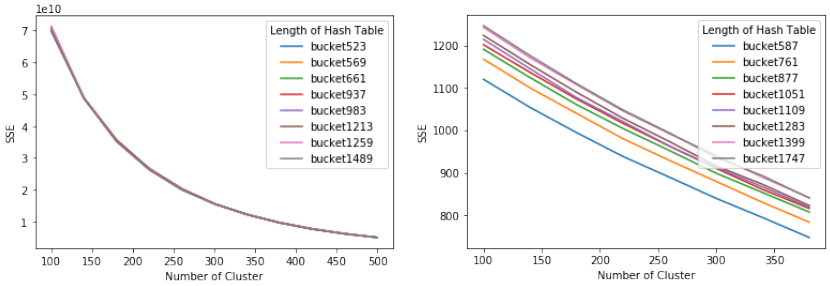

Figure 4: Grid Search for Length of hash table and number of clusters

For normalized vectors, maintaining 587 may be a better choice. For unnormalized vectors, different length of vectors did not present significant distinction, so I stick with a small value (523) for the sake of computation.

Figure 4 do not show and suitable number of clusters especially for normalized data, because grouping the data into a number close to the number of samples may as well do not cluster them at the very first place. With that being said, my ultimate goal is to find the most similar repositories for a given one, and clustering only serves as an exploratory study. Thus, I chose the number of clusters to be 150 and to see if them can beat LDA method with the same setting.

Results

LDA results

Aftering cluster the data into 150 different groups with *KMeans*, I subset the records also in the evaluation set. Define the purity of each group to be the ratio of largest number of single type assigned into this group. Since most clusters in evaluation set may contain only one elements, making the homogeneity and completeness of the results unreasonaly high, I treat recall rate as the mean purity for each groud truth label with menbers more than one and precision to the mean purity for each predicted label with more than one menbers. Clustering result is in Table 1.

Recall	Precision	Homogeneity	Completeness
0.07	0.56	0.800	0.738

Table 1: Results for LDA

Since the recall rate here is to mainly test whether reporsitories sharing many duplicated files have been clustered together, the result seems to be shaky. What is worse, LDA is really bad at running speed with such a giant dataset. Thus, some other methods are required.

Hash Method Result

Similarily, group the collection of repositories into 150 clusters with *KMeans*. The result is shown in Table **??**

	Recall	Precision	Homogeneity	Completeness
unnormalized	0.44	0.51	0.526	0.767
normalized	0.15	0.47	0.789	0.723

Table 2: Results for Hash method both for normalized and unnormalized vectors

In case of both recall rate and precision, unnormalized vectors outperform normalized vectors, which is counter intuitive, not to mention the wierd plot during grid search (which is expected to be an elbow plot like the left hand side in Figure 4). Remember the TF-IDF problem. TF-IDF embedding reduces the effect of document frequency from term frequency, while normalization reduce the effect of scale of vectors. One assumption is that repository size is also a significant feature to determine the similarities of repositories. With that being said, the evaluation set itself is quite subjective. The results just indicate an inclination.

Furthermore, I calculate the homongeneity and completeness of results from LDA and Hash method with unnormalized vectors, being 0.6099 and 0.5048 respectively, meaning that the two methods are somewhat in accordance.

Include Comments information.

Finally, I combined the comments from source code into the identifier names, and did the same above process all over again. The results is given as Table **??**

	Recall	Precision	Homogeneity	Completeness
unnormalized hash	0.44	0.45	0.510	0.754
normalized hash	0.10	0.51	0.810	0.724
LDA	0.07	0.54	0.869	0.731

Table 3: Results for combination of identifiers and comments data

In terms of recall and precision, including comments seems to underperform slightly only containing identifiers.

Conclusion

In conclusion, the project did an exploration of clustering with the identifiers and comments in source code files. The main drawback is lacking of ground truth evaluation. However, I intentionally include some repositories with similar contents. Then used unnormalized vectors from hash methods and return the top similar repositories with Pearson Correlation. It turns out the tested repositories are discovered correctly. Perhaps more information should be extracted in future work, and should scrape repositories of same topic intentionally for evaluation purpose.

Bibliography

C. V. Lopes, P. Maj, P. Martins, V. Saini, D. Yang, J. Zitny, H. Sajnani, and J. Vitek. Déjàvu: a map of code duplicates on github. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):84, 2017. [p1]

V. Markovtsev and E. Kant. Topic modeling of public repositories at scale using names in source code. *arXiv preprint arXiv:1704.00135*, 2017. [p1]

Jingci Wang
Northeastern University
Khoury College of Computer Sciences
Data Science
wang.jingc@husky.neu.edu