Network Science 2017
Final Project

B02901026 莊佾霖
B02901119 梁永勤

June 26, 2017

GitHub[1] launched a contest in 2009[2], which aimed to provide a recommendation system on its website. However, due to some unknown reason, GitHub has not yet provided such service. Hence in this project, we try to build a recommendation system using data of GitHub.
Our project can be categorized into two parts. In the first part, we try to construct a objective network using the data obtained on GitHub Archive. We wish to turn the original bipartite graph, where the two disjoint sets are users and repositories, into two simple graphs, each connecting only users or repositories. We also discuss the obstacles we met during this process, and state our solution for this problem.The second part of the project is link prediction. We aim to produce predictions using an algorithm found on the Internet, and estimate the result for this algorithm. In the end, a seemingly promising approach of link prediction is proposed, and positive feedback was gained. All code can be found at repository frankyjuang/Octomender[3].

1 Introduction

1.1 GitHub Network

GitHub is a web-based Git, or version control repository hosting service, which offers distributed version control and source code management. Individuals on GitHub can be categorized into two types, i.e. users and repositories. By considering the interactions between individuals, we can construct networks according to different features.

In this project, we focus on GitHub data obtained from GitHub Archive[4], then try to analysis and further exploit information contained within the data.

1.2 Link Prediction

Link prediction is a kind of link analysis, focusing on the tendency of edge formation within network. By analyzing the edge distribution, we can infer that particular pairs of vertices possess high probability of forming new links, thus predicting the edge development.

A similar concept, recommendation system, is often mentioned along with link prediction. While they can have similar application, recommendation system and link prediction are different in the way they are defined. Recommendation system aims to provide a target, e.g. a costumer, with elements, e.g. product, that one would possibly be fond of. This is often done by analyzing properties of vertices within network, e.g. mutual similarity, and by trying to relate similar vertices together.

The NetworkX[5] Python library is used for graph operations, and all mathematical and statistical calculations use the NumPy[6] and SciPy[7] libraries. For parallel computing, OpenMP[8] is used.

2 Data and Graph construction

2.1 Data Usage

Data in GitHub Archive is stored in .json file[9], where each packet is based on a single event happened on GitHub, e.g. when a user starred a repository. These events represents the interaction between users and repositories, thus serve as some accordance when constructing the graph. In this project, we make use of two types of events, i.e. `MemberEvent` and `WatchEvent`. Note that `WatchEvent` is related to starring a repository, not watching, and will be denoted as "star" in the following paragraphs. These two events both present user-repo interaction while possessing different magnitude. We can exploit the nature of data to easily form a bipartite graph, with disjoint sets consisting of users and repositories.

To further convert it into unipartite graphs, we conduct the following measures. First, we form an edge between a pair of elements whenever they have at least a common neighbor. Note that the two elements have to be of same type in order to form a unipartite graph. Second, we perform filtering on the newly formed edges, since the graph is likely to be a complete graph if we leave all edges untouched. To do so, we calculate the multiplicity of edges between each pair of elements, leave those with top 10% multiplicity and remove the rest of them. After filtering, we get two unipartite graphs for users and repositories, and is ready to perform similarity metrics on the newly constructed graphs.

2.2 Problem

After applying similarity metrics on the graphs, the only quantities we could obtain is either 0 or 1. This indicates that the graph might consist of disjoint cliques. To verify this, we calculate the proportion of edges with different multiplicities. The result is listed in Table 1.

multiplicity	2016-06-01 member-repo	2016-06-01 member-user	2016-05 member-repo	2016-05 member-user
1	0.1044	0.9222	0.5656	0.9147
2	0.8250	0.0537	0.4084	0.0604

Table 1: The proportion of edges with multiplicity 1 and 2

From Table 1, we can see that most of the edges before filtering has multiplicities between 1 and 2. In this case, we would not be able to tell which edges are more crucial to the network. Without better idea on how to signify importance of edges, we turn back to bipartite graph and search for different algorithms on bipartite link prediction.

2.3 Basic Analysis

Below we provide three degree distribution of our bipartite networks:

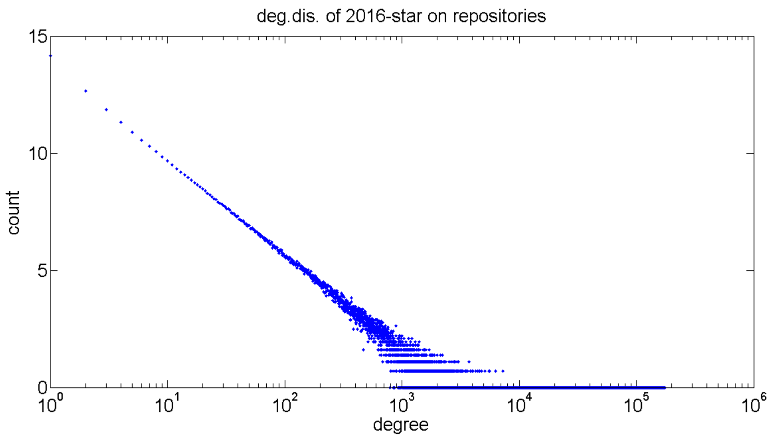
- 2016-star network
- 2017-05-star network
- 2017-05-member network

2.3.1 2016-star Network

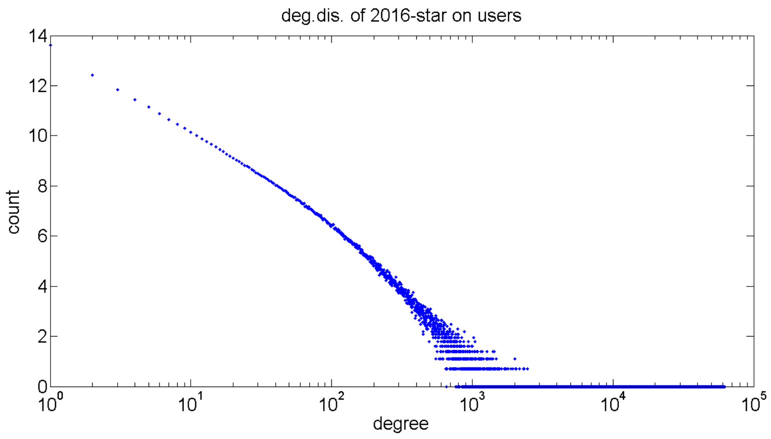
This data records star events throughout 2016. The results are provided in Table 2 and Figure 1.

	Users	Repos	Edges
Number	1,936,247	2,338,123	25,648,267
Avg. Degree	13.246	10.970	-
Max. Degree	62,356	175,225	-
Min. Degree	1	1	-

Table 2: Basic analysis of GitHub 2016-star network



(a) Degree distribution of 2016-star network on repositories



(b) Degree distribution of 2016-star network on users

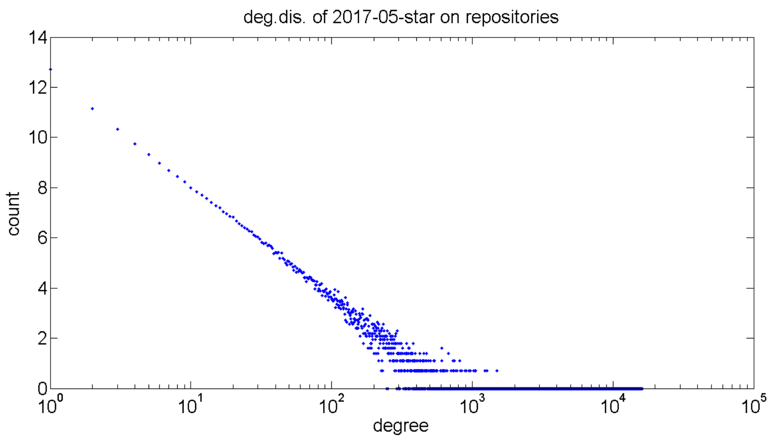
Figure 1: Degree distribution of 2016-star network

2.3.2 2017-05-star Network

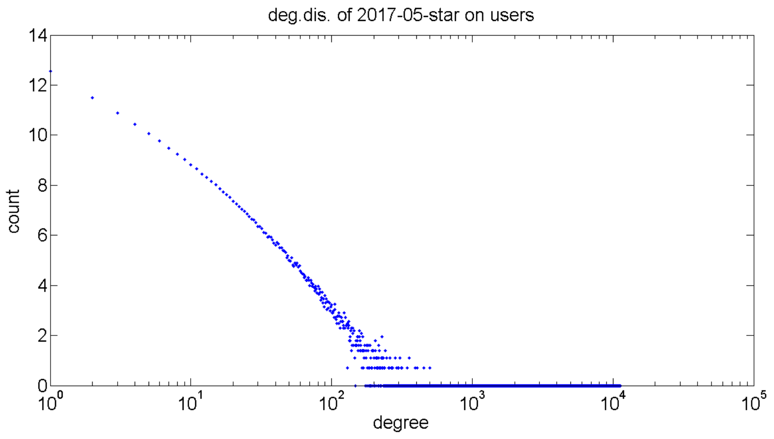
This data records star events in May, 2017. The results are provided in Table 3 and Figure 2.

	Users	Repos	Edges
Number	596,865	518,205	2,644,233
Avg. Degree	4.430	5.103	-
Max. Degree	11,282	16,166	-
Min. Degree	1	1	-

Table 3: Basic analysis of GitHub 2017-05-star network



(a) Degree distribution of 2017-05-star network on repositories



(b) Degree distribution of 2017-05-star network on users

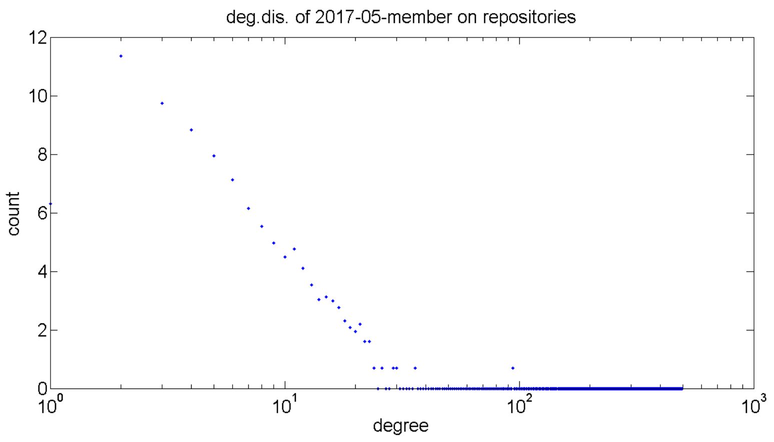
Figure 2: Degree distribution of 2017-05-star network

2.3.3 2017-05-member Network

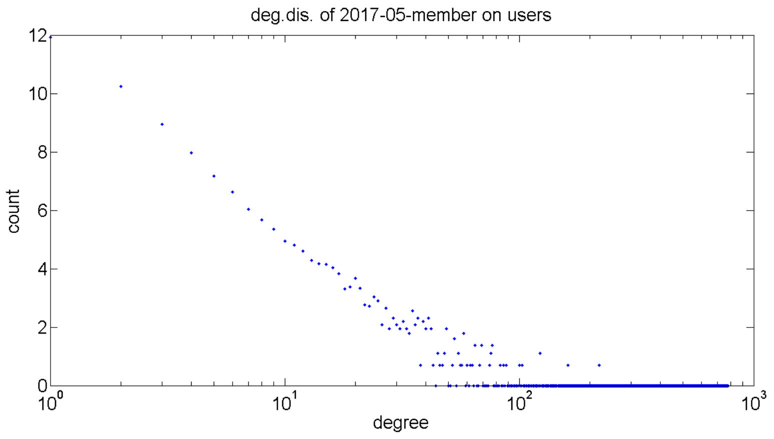
This data records member events in May, 2017. The results are provided in Table 4 and Figure 3.

	Users	Repos	Edges
Number	191,842	115,643	285,856
Avg. Degree	1.490	2.472	-
Max. Degree	779	495	-
Min. Degree	1	1	-

Table 4: Basic analysis of GitHub 2017-05-member network



(a) Degree distribution of 2017-05-member network on repositories



(b) Degree distribution of 2017-05-member network on users

Figure 3: Degree distribution of 2017-05-member network

3 Link Prediction

3.1 Algorithm

As we mentioned above, we failed to construct simple graphs for GitHub network, and usual metrics for unipartite graph would not apply. Hence, we searched through Internet and found a paper[10] that claims to resolve link prediction on bipartite graphs.

The main approach in this algorithm is to analyze the "biadjacency matrix", which is similar to the common adjacency matrix, but that the biadjacency matrix $B \in \left\{0,1\right\}^{|\mathcal{V}|\times|\mathcal{W}|}$ represents the inter-connection between two disjoint sets $\mathcal{V}$ and $\mathcal{W}$. We then apply singular value decomposition (SVD) on the biadjacency matrix, i.e. $B = U\Sigma V$. Here, each column pair of U and V forms a contribution to matrix B, and the corresponding element in Σ represents the weight of this contribution. We exploit this feature and modify Σ such that it gives link prediction scores that signifies the tendency of the edge formation.

3.2 Odd Pseudo Kernel

As mentioned above, we try to signify the weight of those components that show the tendency of the edge formation. Usually, the weighting function should fulfill the following requirements:

- The link prediction score should be higher if the vertices are connected with *many* paths.
- The link prediction score should be higher if the paths are *short*.

Since the graph is bipartite, only the paths with odd length matters. One of the methods proposed in reference[10] is the altered form of matrix exponential, which is often used in unipartite link prediction. The matrix exponential is the sum

$$exp(\alpha A) = \sum_{i=0}^{\infty} \frac{\alpha^i}{i!} A^i,$$

where α is a parameter to be tuned. By keeping the odd components, we thus derive the matrix hyperbolic sine

$$sinh(\alpha A) = \sum_{i=0}^{\infty} \frac{\alpha^{2i+1}}{(2i+1)!} A^{2i+1}.$$

Another proposed method is the rank reduction. This method simply chooses top k singular values and discards the rest of them. The comparison between two methods can be seen in the result below.

3.3 Result

In general, link prediction aims to find out edges that are likely to form in future development. However, this sets an obstacle on the evaluation of result, since no one knows the exact development of the future of a graph. Hence in this section, we evaluate the result by removing certain amount of edges beforehand and "predict" the edges to see if those removed could be found again. We compare two sets, i.e. the "removed" set and the "prediction" set, and calculate the proportion of intersection between two sets. the results are provided as follow.

K	Kernel	Accuracy(%)
10	sinh	1.7191
	rr	5.6186

Table 5: Accuracy of predicting removed edges of 2016-star network

K	Kernel	Accuracy(%)	K	Kernel	Accuracy(%)
10	sinh	2.8588	300	sinh	2.8988
	rr	2.6126		rr	2.8420
30	sinh	3.1794	700	sinh	2.6863
	rr	3.1037		rr	2.4026
70	sinh	2.8004	1500	sinh	2.8193
	rr	3.0653		rr	2.2327
150	sinh	3.0057	3000	sinh	2.1743
	rr	3.2514		rr	2.2500

Table 6: Accuracy of predicting removed edges of 2017-05-star network

K	Kernel	Accuracy(%)	K	Kernel	Accuracy(%)
10	sinh	0.8757	300	sinh	5.9649
	rr	0.8757		rr	4.7368
30	sinh	3.1579	700	sinh	8.0986
	rr	1.5789		rr	5.8099
70	sinh	3.3275	1500	sinh	9.1068
	rr	3.1524		rr	6.4799
150	sinh	7.3555	3000	sinh	10.3873
	rr	4.7285		rr	8.9789

Table 7: Accuracy of predicting removed edges of 2017-05-member network

3.4 Discussion

From Table 5, 6 and 7, we see that most of the time, precision of the result is below 10%. This is pretty low, both intuitively and compared to the result in reference[10]. There are several possibilities to this:

- The computation complexity of truncated SVD is $O(k^2n)$, where $n = \min(|\mathcal{V}|, |\mathcal{W}|)$, and $\mathcal{V}$, $\mathcal{W}$ are the disjoint sets of bipartite graph. As k, i.e. the dimension of SVD, grows higher, both space and time grows drastically. The highest k we could implement is $k = 3000$, which sets a limit on the resolution of our model.
- We use accuracy of predicting removed edges as the evaluation of result. However, this could cause false negative, since only the "edges that has already formed and temporarily removed" is considered correct, while the "edges that should but have not yet formed" would cause error. This is due to the random removal applied here, which can be solved by removing according to timeline.

Another view point is observation between hyperbolic sine and rank reduction method. Since the library used requires parameter k to be set, this is equivalent to always performing rank reduction. The results in Table 5, 6 and 7 confirmed that hyperbolic sine method after rank reduction does better than rank reduction alone.

4 Alternative Solution

After the final demonstration, we came up with an new solution to this problem.

Instead of forming a unipartite graph for each disjoint set, we directly calculate similarities among vertices. Since all vertices in the same subset have the same neighbor set, e.g. all users in "user set" connect to the same "repository set", we can still calculate the similarity among vertices of the same side, while circumventing the difficulty of forming new graphs. Also, by specifying a target user, we can locally search for the related new repositories according to the user's taste, decreasing unrelated factors. The procedure is provided in the following Table 8. We use $\mathcal{U}$ to denote "user set" and $\mathcal{R}$ to denote "repository set", while $\Gamma(s)$ denotes the set of neighbors of s and $\Gamma(\mathcal{S}) = \big\{\Gamma(s_1) \cup \Gamma(s_2) \cup \ldots \cup \Gamma(s_i) \cup \ldots \cup \Gamma(s_n) : s_i \in \mathcal{S}\big\}$.

```
Input: target user $t \in \mathcal{U}$, edge list $\mathcal{E} \in \{(u,r) : u \in \mathcal{U}$ and $r \in \mathcal{R}\}$, score threshold $s_\tau$
$\mathcal{R}_t \leftarrow \Gamma(t)$
$\mathcal{R}_p \leftarrow \Gamma(\Gamma(\mathcal{R}_t)) \setminus \mathcal{R}_t$
$s(r_p) \leftarrow 0, \forall r_p \in \mathcal{R}_p$
for each pair $(r_t, r_p) \in (\mathcal{R}_t, \mathcal{R}_p)$
    Calculate similarity metric $sim(r_t, r_p)$
    $s(r_p) \leftarrow s(r_p) + sim(r_t, r_p)$
end for
Output: $\mathcal{R}_s = \{r : r \in \mathcal{R}_p$ and $s(r) \geq s_\tau\}$
```

Table 8: An alternative approach for link prediction

The similarity metric used in Table 8 is chosen from reference[11], i.e. Jaccard's coefficient and Adamic coefficient. We realized a preliminary version of this approach and generated two sets of prediction result (Table 9 and Table 10) for an active GitHub user `leomao` (`https://github.com/leomao`).

Rank	Repository	Score
1	KaimingHe/deep-residual-networks	132.6990
2	carpedm20/deep-rl-tensorflow	124.0000
3	matthiasplappert/keras-rl	122.3100
4	dennybritz/reinforcement-learning	119.0300
5	ibab/tensorflow-wavenet	118.4670
6	tensorflow/models	115.1740
7	tflearn/tflearn	115.0270
8	jtoy/awesome-tensorflow	108.2450
9	blei-lab/edward	101.5530
10	carpedm20/DCGAN-tensorflow	99.8124

Table 9: Top 10 prediction for `leomao` of 2016-star network (Set 1)

Rank	Repository	Score
1	tensorflow/models	1159.0000
2	firehol/netdata	1026.8800
3	naptha/tesseract.js	999.1720
4	facebook/draft-js	994.2890
5	jtoy/awesome-tensorflow	945.8900
6	tflearn/tflearn	896.7480
7	airbnb/superset	895.4480
8	FormidableLabs/webpack-dashboard	886.1290
9	mzabriskie/axios	849.2210
10	Microsoft/CNTK	834.5890

Table 10: Top 10 prediction for `leomao` of 2016-star network (Set 2)

The difference between Set 1 and Set 2 is that in Set 1 only neighbors of target user whose degree is less than or equal to 1000 are considered, and the ones with degree less than or equal to 5000 in Set 2. That is, $\mathcal{R}_t \leftarrow \{r : deg(r) \leq 1000, r \in \Gamma(t)\}$ for Set 1. According to `leomao`, `leomao` would like to star all of these repositories. Furthermore, Set 2 provides more diversity than Set 1, appearing to have more charm.

5 Conclusion and Future Perspectives

Up to now, GitHub doesn't offer any recommendation service, so we aimed to come up with an effective recommender system based on user-repo networks. On one hand, a proposed algorithm dealing with biadjacency matrix was tested on predicting removed links, but it turned out to be unproductive, and several negative factors were discussed. On the other hand, we offered an alternative solution, targeting recommendation service from a local aspect. Though the result of it seemed promising, we lacked an objective evaluation method. Other issues such as prevention from being dominated by extremely high degree repositories also awaits more robust solutions in the future.

References

[1] Github. `https://github.com`

[2] GitHub contest in 2009. `https://github.com/blog/466-the-2009-github-contest`

[3] Y. L. Juang, Y. C. Liang. Github Repo Recommender System. 2017 Network Science Final Project. `https://github.com/frankyjuang/Octomender`

[4] GitHub Archive. `https://www.githubarchive.org`

[5] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, "Exploring network structure, dynamics, and function using NetworkX" , in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008

[6] Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37

[7] Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, `http://www.scipy.org/`

[8] OpenMP Architecture Review Board. OpenMP Application Program Interface Version 4.0. 2008. `http://www.openmp.org/wp-content/uploads/OpenMP4.0.0.pdf`

[9] GitHub Archive Event type.
`https://developer.github.com/v3/activity/events/types`

[10] J. Kunegis, E. W. De Luca, S. Albayrak. "The Link Prediction Problem in Bipartite Networks". arXiv: 1006.5367, 2010
`https://arxiv.org/pdf/1006.5367.pdf`

[11] Pair-wise link prediction metric. `http://be.amazd.com/link-prediction`