# Hot or Not? Finding Star Projects in Coding Social Network

Ming-Ju Valentine Lin, Sining Ma, Masoud Tavazoei
mjvallin@stanford.edu, sma87@stanford.edu, tavazoei@stanford.edu

***Abstract*—Open-source software and social coding are in demand these days. How would an open-source software stand out and engage more users (development, usage, subscription, etc.) than others? In this paper, we first try to understand the structure of social coding network and then we will attempt to classify popular projects on GitHub using degree centrality, PageRank and logistic regression. Our data are based on GitHub events: ForkEvent, WatchEvent, IssueCommentEvent, PullRequestEvent, PushEvent, ReleaseEvent, IssuesEvent, MemberEvent, and StatusEvent. We will build project-project networks from four of these events (Fork, Watch, Issue Comment and Pull Request) and run logistic regression algorithm on all of these events. This paper will help project contributors to improve their project popularity by focusing on the determining factors and also help users to identify projects that have the potential to become popular later on, so that they can have a wiser choice for their tech stack.**

***Keywords*—Github, influential, classification, logistic regression, centrality, PageRank**

## I. Introduction

Social coding bring developers together and enables an ecosystem of open source software. Established projects such as those from Apache Foundation are built by many volunteering developers. GitHub, one of the popular repository hosting services allows developers to contribute to common code bases and collaborate in software development. Mostly importantly, GitHub has been used to host many successful and popular projects, for example Homebrew and Docker.

We are interested in how an open-source software becomes popular. We will use empirical graph properties such as degree centrality, PageRank, and degree distribution as well as logistic regression to determine what projects are influential and why.

## II. Related Work

Similar research topics regarding influence measurement have been done by many others. Measuring User Influence in Twitter: The Million Follower Fallacy[1] represents the comparison of three measures of influence: follows, retweets, and mentions. The dynamics of user influence across topics and time are investigated. This paper concludes that retweets are driven by the content of a tweet, mentions are driven by the name of the user, and follows represent a user's popularity.

The "follow" measure which identifies users who get lots of attention from their audience is essentially degree centrality in network analysis. The most mentioned users are mostly celebrities. Across all three measures, the top influentials were generally recognizable public figures and websites. These three top lists have marginal overlaps. The analysis mentioned in this part of the paper is similar to community structure. Based on the analysis in the paper, retweets and mentions are more relevant, but follows do not relate to other measures.

Likewise, Network Structure of Social Coding[2] investigated GitHub project networks as well as developer networks in order to answer: How strong are the relationships among projects and among developers? and What are the most influential projects and developers?

They constructed a graph of prjects by linking pairs of projects that share at least one contributor and similarly the constructed a network of developrs in which two developers were connected if they had collaborated on at least one project. Using empirical graph properties such as PageRank, network diameter, average shortest path and degree centrality, Thung et al. identified influential projects and developers and also found that the diameter and shortest path in both project network as well as developer network are significantly smaller than the corresponding metrics for many other real networks (e.g. Facebook or SourceForge). Therefore, they concluded that GitHub as a social platform for coding is indeed getting projects and developers closer to each other.
In another research, Dabbish et al [4] interviewed 25 rock stars in GitHub world and found out that users infer a lot from other users' actions. Users combine these inferences into effective strategies for coordinating work, advancing technical skills and managing their reputation. So for example they find that number of "Forks" or "Wathces" is a measure of a importance to community and users tend to follow those projects that the community already cares about. Or recency

and volume of activities reflects interest and level of commitment. Their study shows that users' action play a critical role in forming the project's reputation and that is what are going to investigate deeper in this article.

## III. Objective

We first target network empirical properties (degree centrality and pageRank) to list top 10 projects. We cross compare results of each event graph and draw a conclusion on similarities among the top lists. We also look further at some individual top projects to tell if any property is a good technique to predict project popularity and whether or not high-ranked projects are actually well-established open-source software.

Machine learning algorithm is a more complicated and accurate approach to predict whether a GitHub project is popular. The goal is that we plan to predict a project's popularity based on old project event data. If this method works well, we do not need to know pull request count. Neverthless, guess if this project is popular and worths following.

## IV. DataSet

We use GitHub archive data (https://www.githubarchive.org/) as our data source. We collect all public "Fork", "Watch", "Pull Request" and "Comment on Issues" events between 01/01/2015 and 06/01/2015 for our graph analysis. We also discuss about logistic regression to classify which GitHub projects are popular. For this part, we collect data from 01/01/2015 to 12/01/2015. We chose a larger time window to train our predictive model to leverage on all the available data and achieve a high training precision as well as testing precision. As the volume of data is massive, we use Google's BigQuery platform to perform data retrieval and analysis. Google BigQuery is a distributed data platform that allows user to run SQL-like queries against append-only tables, using the infrastructure developed and maintained by Google.

We consider the following Github events:

- ForkEvent
- WatchEvent
- IssueCommentEvent
- PullRequestEvent
- PushEvent
- ReleaseEvent
- IssuesEvent
- MemberEvent
- StatusEvent

For empirical graph properties, we run queries to extract projects from the given date range for four events: (watch, fork, issue comment and pull request). There are in total 1,214,552 projects. For each event, we link two projects together when they have at least one common user who triggered the same type of event. We then use Snap to build four undirected graphs. We load these projects as vertices into the four graphs: Watch Event Graph, Fork Event Graph, Issue Comment Graph, and Pull Request Graph.

Table I summarizes the basic summary statistics for these 4 created networks. Since Watching and forking are easier to do for the users comparing to sending pull request or commenting on issues, it is expected that networks based on "Watch" and "Fork" cover a wider range of projects.

| Graph | Number of Nodes | Number of Connections |
|---|---|---|
| Fork | 554874 | 65577338 |
| Watch | 788727 | 463054873 |
| IssueComment | 238878 | 21647636 |
| PullRequest | 311226 | 33345851 |

TABLE I: Graph Data Size

## V. Discussion

### A. Centrality Measure

Table II, III, IV and IV list the top 10 repositories based on degree for the 4 constructed networks. Although most of the projects that are listed in these 4 tables are well-known projects, there are still some cases, especially for the "Pull Request" network, that some listed projects are not popular or well-established. For example, "anyenv" or "justing-podcast-crawler" are two projects that have made to the top list while they are both small and infamous. Once we look closely at these two projects, it becomes clear that developers of these projects have actively sent pull requests for numerous ad-hoc projects. This is actually one of the inherent problems of using degree as a measure, such that we may wrongly consider spam projects as popular or well-established. This will not be a problem in Page Rank as random teleports solves spamming problem to a great extent.

Table VI, shows the cosine similarity between the list of top 10,000 projects in all 4 networks. There is a high similarity between the top projects in the "Fork" network and the "Pull Request" network. This suggests that once users fork a project to modify their own use cases, it is likely that they will give back to the open source

| Project ID | Project Name | Degree |
|---|---|---|
| 2126244 | twbs/bootstrap | 25158 |
| 29028775 | facebook/react-native | 23832 |
| 460078 | angular/angular.js | 21279 |
| 13491895 | vhf/free-programming-books | 20563 |
| 10270250 | facebook/react | 20511 |
| 12791642 | BVLC/caffe | 19730 |
| 943149 | mbostock/d3 | 19229 |
| 3228505 | atom/atom | 18571 |
| 1300192 | octocat/Spoon-Knife | 18494 |
| 1062897 | github/gitignore | 18285 |

TABLE II: Fork Event Graph Top 10 Degree Centrality

| Project ID | Project Name | Degree |
|---|---|---|
| 29028775 | facebook/react-native | 189559 |
| 33614304 | nvbn/thefuck | 173976 |
| 32484381 | ripienaar/free-for-dev | 161788 |
| 29247444 | yaronn/blessed-contrib | 150934 |
| 9384267 | atom/atom-shell | 144679 |
| 3673976 | sdelements/lets-chat | 143767 |
| 14747598 | typicode/json-server | 141683 |
| 29900673 | phanan/htaccess | 139858 |
| 10270250 | facebook/react | 139450 |
| 30384844 | Flipboard/react-canvas | 138501 |

TABLE III: Watch Event Graph Top 10 Degree Centrality

| Project ID | Project Name | Degree |
|---|---|---|
| 321278 | npm/npm | 12633 |
| 7691631 | docker/docker | 12349 |
| 206084 | Homebrew/homebrew | 11898 |
| 3228505 | atom/atom | 11592 |
| 1420493 | travis-ci/travis-ci | 11435 |
| 27193779 | iojs/io.js | 10444 |
| 3623050 | caskroom/homebrew-cask | 9940 |
| 1272666 | jshint/jshint | 9516 |
| 3678731 | webpack/webpack | 9477 |
| 7548986 | laravel/framework | 9470 |

TABLE IV: Issue Comment Event Graph Top 10 Degree Centrality

| Project ID | Project Name | Degree |
|---|---|---|
| 26467939 | leandromoreira/redlock-rb | 9221 |
| 24157194 | nikolay/anyenv | 9214 |
| 23521385 | nikolay/wshare | 9214 |
| 27442102 | fablab-ka/OpenSCAD2D | 9213 |
| 32447940 | jizhouli/justing-podcast-crawler | 9213 |
| 29251868 | tetsuya00/Dotfiles | 9212 |
| 33564006 | mrkrstphr/illacceptanything | 9163 |
| 24560307 | babel/babel | 8757 |
| 26295345 | dotnet/corefx | 8586 |
| 2126244 | twbs/bootstrap | 8544 |

TABLE V: Pull Request Event Graph Top 10 Degree Centrality

| | Fork | Issue Comment | Pull Request | Watch |
|---|---|---|---|---|
| Fork | 1 | 0.11 | 0.70 | 0.18 |
| Issue Comment | | 1 | 0.06 | 0.13 |
| Pull Request | | | 1 | 0.04 |
| Watch | | | | 1 |

TABLE VI: Cosine Similarity between List of Top 10,000 Nodes with Highest Degree in 4 Networks

community and submit a pull request to share what they have changed with other users.

Tables VII, VIII, IX and X show the list of top 10 projects in the 4 networks based on PageRank. By inspecting all the top projects we can confirm that in all 4 networks the top 10 projects are indeed popular and influential projects. Consequently, as it can be observed from Table XI, the list of top 10,000 projects in all 4 networks have a very significant similarity. These findings confirm that PageRank is a more reliable and robust metric to identify popular projects in GitHub.

| Project ID | Project Name | PageRank * 10K |
|---|---|---|
| 1300192 | octocat/Spoon-Knife | 9.955 |
| 14204342 | jtleek/datasharing | 6.645 |
| 2126244 | twbs/bootstrap | 4.399 |
| 15917132 | rdpeng/ProgrammingAssignment2 | 3.791 |
| 460078 | angular/angular.js | 3.545 |
| 13491895 | vhf/free-programming-books | 3.174 |
| 16599031 | barryclark/jekyll-now | 3.025 |
| 943149 | mbostock/d3 | 2.935 |
| 10270250 | facebook/react | 2.723 |
| 29028775 | facebook/react-native | 2.676 |

TABLE VII: Fork Event Graph Top 10 PageRank

| Project ID | Project Name | PageRank * 10K |
|---|---|---|
| 33614304 | nvbn/thefuck | 3.267 |
| 29028775 | facebook/react-native | 3.185 |
| 32484381 | ripienaar/free-for-dev | 2.913 |
| 14194174 | alex/what-happens-when | 2.405 |
| 10270250 | facebook/react | 2.399 |
| 13491895 | vhf/free-programming-books | 2.318 |
| 29900673 | phanan/htaccess | 2.309 |
| 21737465 | sindresorhus/awesome | 2.252 |
| 9384267 | atom/atom-shell | 2.229 |
| 29247444 | yaronn/blessed-contrib | 2.211 |

TABLE VIII: Watch Event Graph Top 10 PageRank

Figures 1-4 depict the degree distribution for all 4 networks in log-log scale. As it can be seen the degree distribution of "Fork" and "Watch" networks are almost linear so it seems that "Fork" and "Watch" networks are formed in a preferential attachment process.

### *B. Logistic regression*

We run queries to retrieve the count of the above events group by project id, sort the project ids based

| Project ID | Project Name | PageRank * 10K |
|---|---|---|
| 7691631 | docker/docker | 9.628 |
| 206084 | Homebrew/homebrew | 8.659 |
| 3228505 | atom/atom | 8.384 |
| 3470471 | FortAwesome/Font-Awesome | 8.330 |
| 321278 | npm/npm | 7.400 |
| 1420493 | travis-ci/travis-ci | 7.309 |
| 27193779 | iojs/io.js | 6.002 |
| 10483981 | isaacs/github | 5.253 |
| 8514 | rails/rails | 5.051 |
| 481872 | mitchellh/vagrant | 4.664 |

TABLE IX: Issue Comment Event Graph Top 10 PageRank

| Project ID | Project Name | PageRank * 10K |
|---|---|---|
| 24392063 | udacity/create-your-own-adventure | 10.767 |
| 206084 | Homebrew/homebrew | 6.994 |
| 1300192 | octocat/Spoon-Knife | 5.846 |
| 24659289 | LarryMad/recipes | 4.627 |
| 3623050 | caskroom/homebrew-cask | 4.158 |
| 8514 | rails/rails | 3.827 |
| 20413356 | deadlyvipers/dojo_rules | 3.100 |
| 15783450 | jlord/patchwork | 2.878 |
| 33564006 | mrkrstphr/illacceptanything | 2.873 |
| 724712 | rust-lang/rust | 2.798 |

TABLE X: Pull Request Event Graph Top 10 PageRank

| | Fork | Issue Comment | Pull Request | Watch |
|---|---|---|---|---|
| Fork | 1 | 0.25 | 0.29 | 0.55 |
| Issue Comment | | 1 | 0.30 | 0.0.28 |
| Pull Request | | | 1 | 0.24 |
| Watch | | | | 1 |

TABLE XI: Cosine Similarity between List of Top 10,000 Nodes with Highest PageRank in 4 Networks

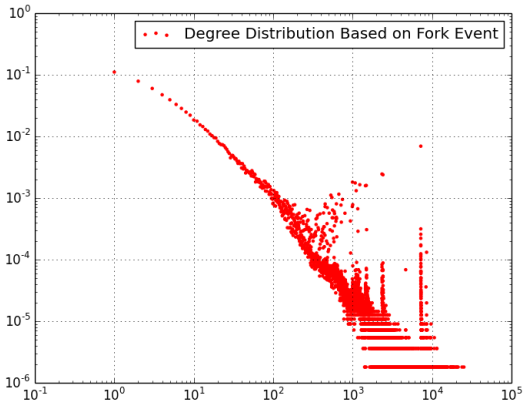


Fig. 1: Degree Distribution Based on Fork Event Network

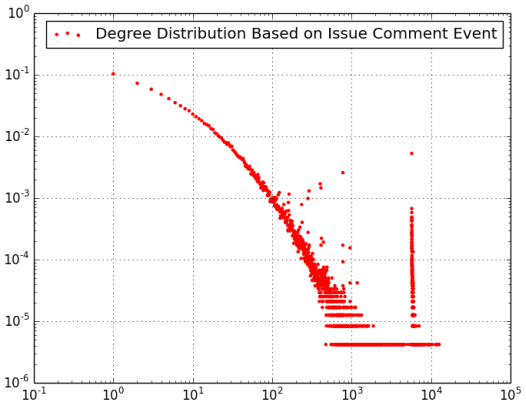


Fig. 2: Degree Distribution Based on Issue Comment Event Network

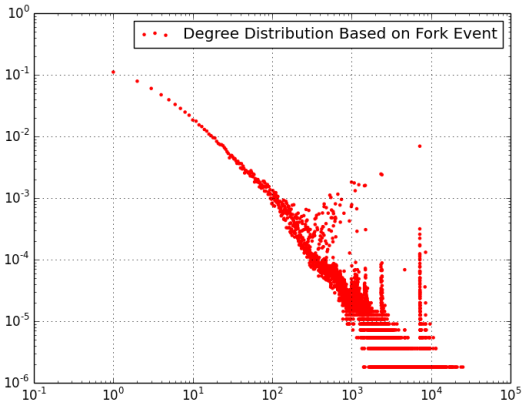


Fig. 3: Degree Distribution Based on Fork Event Network

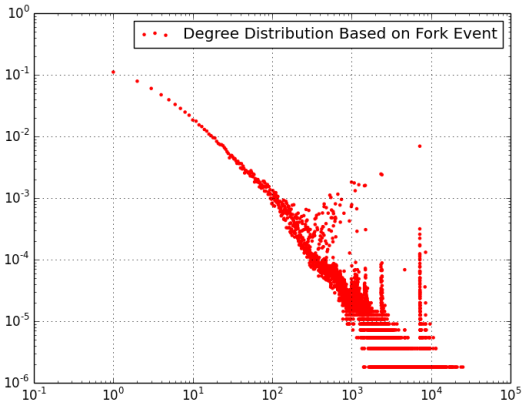


Fig. 4: Degree Distribution Based on Watch Event Network

on the number of pull requests and label the top 10% as influential projects and the remaining projects as non-influential projects. This indicates that labeling the top 10% projects as positive ("1"), and the other 90% as negative ("0"). Next, we split these data into two parts: 70% of the data are used as training data and 30% data as testing data. At first, we treat all feature events as training data. However, when we clean and merge these training data, we see that most projects IssueEvent counts are zero. This means that this feature is not an important factor to decide whether or not a project is influential. For each project, we check if it is an influential project in previous labelled data. Finally, we choose six features and labelled project results applying online logistic regression algorithm provided by Apache Mahout.

## VI. Modeling

The purpose of logistic regression is to identify influential and non-influential projects. This is a binary classification problem. Logistic regression is a widely used classification algorithm for this purpose. We propose to use logistic regression with sigmoid function. Mahout implementation uses Stochastic Gradient Descent (SGD) to large training datasets. This rule with the following equation:

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

$$\text{where} \quad h_\theta(x) = g(\theta^{\mathrm{T}}x) = \frac{1}{1 + e^{-\theta^{\mathrm{T}}x}}$$

For the learning rate $\alpha$, the implementation of this algorithm plans to try a fixed learning rate value. The equation of stochastic gradient ascent rule derives from taking derivative of logarithmic likelihood function.

$$l(\theta) = \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$$

We measure both training precision and testing precision. If the model is overfitted, a regularizer should be added to avoid this problem. We plan to apply both L1 and L2 regularization. The stochastic gradient ascent is modified as:
L1:

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)} + \frac{\lambda}{m} sign(\theta_j)$$

L2:

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)} + \frac{\lambda}{m}\theta_j$$

Here, $m$ is the total number of projects. $\lambda$ is the regulariation tuning parameter.
When online logistic regression model is trained based on our given dataset, training dataset and testing dataset are tested to predict if one project is popular or unpopular. This prediction is compared with the correct result. We summarize the correct prediction count and divide by total events count to calculate training and testing precision.

## VII. Results

### A. Logistic Regression

We tried both time range from 01/01/2015 to 06/01/2015 and time range from 01/01/2015 to 12/01/2015. Applying the default regularization parameter, logistic regression algorithm performs better when more training data are used in this classification model. Table XII shows the difference between two time ranges.

| Time range | Training precision | Testing precision |
|---|---|---|
| 01/01/2015-06/01/2015 | 0.707 | 0.710 |
| 01/01/2015-12/01/2015 | 0.838 | 0.842 |

TABLE XII: Two time range results

It is clear that Mahout online logistic learning algorithm performs better when more training data are utilized.
After we decide to consider a wider time range from 01/01/2015 to 12/01/2015, the next step is tuning training parameters. Referring to the website suggestion [3], we start with $\alpha = 1, \lambda = 3.0e^{-5}$, step offset is 1000 and decay exponential is 0.9. Table XIII describes the results of tuning regularization parameters $\lambda$. The result shows that we do not encounter an overfit problem. The algorithm testing precision is always better than training precision. 0.916 is a nice result. We also tried some other method to improve this algorithm. We manually add more training examples by adding 10 more for loop to apply the same one set of training data to train this training model. Table XIV shows the results of running

| | Training precision | Testing precision |
|---|---|---|
| $\lambda = 3.0e^{-5}$ | 0.913 | 0.916 |
| $\lambda = 1.0e^{-5}$ | 0.913 | 0.916 |
| $\lambda = 1.0$ | 0.736 | 0.739 |
| $\lambda = 10.0$ | 0.824 | 0.830 |

TABLE XIII: Tuning regularization parameter $\lambda$

| Loop Count | Training precision | Testing precision |
|---|---|---|
| 1 | 0.913 | 0.916 |
| 10 | 0.914 | 0.917 |

TABLE XIV: For loop count to improve algorithm

the entire set of training data once or 10 times when we fix all tuning parameters, $\lambda, \alpha$. It indicates that this approach can hardly improves the result.
L1 regularization and L2 regularization result is very similar according our experiment for this data set. Training precision 0.914 and testing accuracy 0.917 are the best result.

## VIII. Conclusion

We calculated node degree centrality and PageRank as two measures of centrality for the 4 networks that we created based on "Fork", "Watch", "Issue Comments" and "Pull Request" events. We found that using degree centrality can lead to spurious list of top projects. This is mainly due to degree centrality measure cannot handle farming structures and may mistakenly identify some spam project as important ones. However PageRank has proved to be a more robust metric and resulted in identifying actually popular and well-established projects and consequently the lists of top projects with highest Page Rank in all 4 networks exhibit a high similarity. Moreover the degree distribution for "Fork" and "Watch" networks follow a power law distribution. This suggests that users tend to watch or fork projects that are already popular and of which many users have already watched or forked. This finding is consistent with the finding in "Social Coding in GitHub" [4] which claims that users find interesting projects by looking at the number of "Watches" and "Forks".

Logistic regression results match our expectation that connections between feature events and project popularity are highly related. It also shows that not all feature events are as important as others. Issue event is an example. We think that this event should describe whether or not a project is followed by many contributors at first but data and learning results tell us this event is not relevant. Furthermore, regularization improves the learning algorithm. It avoids overfitting and improves learning precision. Increasing the amount of training data solves underfittering problem. Feature selection, training data quantity and regularization parameter tuning are three essential factors to make machine learning perform better.

## IX. Future

- It could be worthwhile to further study top projects calculated by both degree centrality and PageRank to see if the number of commits, size of commits, number of contributors and type of contributors play a role in making a project popular.
- We can look at contributors and users of top projects to study the social/cultural aspect of these open-source projects. For instance, it is suggested by [6] that American companies use open-source software to save cost while European companies like the flexibility of open-source software.
- Many other classification algorithms are discussed and applied in industry and academic papers. Naive Bayesian, SVM or decision trees are widely applied. As one of the future work, we can try these solutions to discover if some of these algorithms can improve final results. We can also switch algorithm category to ranking algorithm. Ranking algorithnm can sort all projects popularity according to given features. Boosting tree or random forest algorithms are widely used.
- We choose some basic training features based on GitHub archive dataset. More features can be applied in our feature work. PageRank as one of graph importance factor is an ideal feature should be considered. Other exmaples are doing mathematical operation against these existing features, such as calculating square of some event count.

## References

[1] Cha, Meeyoung, Hamed Haddadi, Fabricio Benevenuto, and P. Krishna Gummadi. "Measuring User Influence in Twitter: The Million Follower Fallacy."ICWSM 10, no. 10-17 (2010): 30.

[2] Thung, Ferdian, Tegawendé F. Bissyandé, Daniel Lo, and Lingxiao Jiang. "Network structure of social coding in github." In Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on, pp. 323-326. IEEE, 2013.

[3] Mihai Costin. Tame the Machine Learning Beast with Apache Mahout.
(http://www.3pillarglobal.com/insights/how-to-tame-the-machine-learning-beast-with-apache-mahout)

[4] Dabbish, Laura, et al. "Social coding in GitHub: transparency and collaboration in an open software repository." Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work. ACM, 2012.

[5] Thung, Ferdian; LO, David; and JIANG, Lingxiao, "Network Structure of Social Coding in GitHub" (2013). Research Collection School of Information Systems.

[6] Paul, Ryan. "Surveys show open source popularity on the rise in industry". Jan 20, 2006. *Ars Technica*. [Online]. Available: http://libguides.murdoch.edu.au/c.php?g=246207&p=1640694. [Accessed Dec. 7, 2015].