NAME: Nicholas Hanoian
DATE: May 4, 2020
Homework: Final Project Review
STAT/CS 387

1 Reproducing Klug and Bagrow (2016)

We have begun work on the project by attempting to reproduce the original results of Klug and Bagrow. We will refer to this paper by Klug and Bagrow as the original paper within this document. The raw data is stored in gzipped json lines files with records of all events on the public GitHub API. There is one file for each hour, and we are currently only looking at the original data window from Klug and Bagrow which is between January 1, 2013 and April 1, 2014. There are three main parts to this process:

1. Aggregate the raw data from the json lines files into a more usable format for analysis.
2. Filter out teams which do not satisfy the constraints set up by the original paper.
3. Use the aggregated, filtered data to compute the values which are presented in the figures from the original paper and plot them.

We have created a GitHub repository within the Baglab organization to organize the code used to do these three steps. The repository is located at https://github.com/baglab/github-burstiness (requires membership of the Baglab GitHub organization to access). In the following subsections, we will describe our efforts and findings for each of these three key parts.

1.1 Aggregating data from raw files

The first step is to take the chronologically ordered json lines files containing all events from the GitHub API and transform it into a more useful format. There is approximately 19.1 GB of raw data in the original data window, and 810 GB in total on the Xenif server (February 2011–November 2019).

This script to aggregate the data currently takes slightly less than an hour to run on the Xenif server. If we assume that this aggregation scales linearly with bytes of data, we would expect the aggregation of all available data to take around 42.4 hours. This is not necessarily a valid assumption—it may take much longer than this. We may need to put some work into speeding up the script in some way. In addition, it is likely that running the existing script on all of the data would cause us to run out of memory. The scripts may need to be adjusted for this as well, or run on a different machine.

Our current method takes the raw data and aggregates it into three tabular files:

1. `pushes.csv`: timestamp, username, repository ID, size of push, and location of user for all PushEvents
2. `team_members.csv`: repository ID and username for all people on teams. Note: Both repository IDs and users can appear more than once because teams can have many users and users can be on many teams.

3. `repository_stats.csv`: repository ID, current number of stargazers, maximum number of stargazers, size of repository, number of watchers, number of forks, and URL for each repository.

Our current method for extracting data from each event is as follows:

1. If the event has a repository attribute, Update the statistics that we are storing about that repository (stargazers, etc.).
2. If it is a PushEvent, add it to the list of PushEvents.
3. If the event is a PushEvent, CreateEvent, or PublicEvent, add the user who initiated the event to that repository's team, as all three require permission to write to the repository.
4. Update counts of event types.

We are keeping track of the number of occurrences each type of event, separated by whether or not the event has a repository attribute. This is because we completely ignore any event which does not have a repository attribute, and we want to be sure that we are not missing anything. The counts of events which had repository attributes is shown below in Table 1, and the counts of events which did not have repository attributes is shown below in Table 2. The 7,916 PushEvents which do not have repository attributes are concerning, but they are likely from pushes to Gists (single-file projects) as opposed to repositories.

Event Type	Count
PushEvent	55 431 789
CreateEvent	12 390 256
WatchEvent	10 125 324
IssueCommentEvent	9 868 009
IssuesEvent	6 173 753
PullRequestEvent	4 367 704
ForkEvent	4 079 024
GollumEvent	1 787 663
DeleteEvent	1 404 365
PullRequestReviewCommentEvent	1 112 947
PublicEvent	101 494
CommitCommentEvent	969 515
MemberEvent	594 340
ReleaseEvent	147 466
TeamAddEvent	31 589
DownloadEvent	8806

Table 1: Number of events of each type which did have repository attributes.

Event Type	Count
FollowEvent	1 624 674
GistEvent	1 191 410
CreateEvent	83 591
PushEvent	7916
IssuesEvent	7211
PullRequestEvent	1032
DeleteEvent	390
ReleaseEvent	151
PullRequestReviewCommentEvent	120
MemberEvent	86
IssueCommentEvent	63
CommitCommentEvent	45
TeamAddEvent	24
WatchEvent	17
PublicEvent	4
GollumEvent	1
ForkEvent	1

Table 2: Number of events of each type which did not have repository attributes.

1.1.1 Ambiguities

There are several ambiguities regarding the aggregation of data. From the original paper alone, it is not clear which event types are considered and which are not.

It is not clear whether we should be considering CreateEvents (creating of branches or tags) and PublicEvents (a repository changing from private to public) should be considered or not. In addition, there is another event type called a MemberEvent which is when a user accepts an invitation to join a repository. This would signify that the user is a primary member of that team.

The original paper also mentions that 67.8 percent of the repositories were formed within the data window. There is an event type called RepositoryEvent which is supposed to be triggered by the creation of a new repository, but we did not find any records of this type of event within the original data window (see Tables 1 and 2). It is not clear how this statistic was calculated for the original paper.

It is also not clear whether April 1, 2014 is included within the data window. We feel that it would make more sense to only consider the full months of January 2013 through the end of March 2014, but the paper say that they considered data from January 1, 2013 to April 1, 2014. As of now, we are including April 1 within our analyses.

1.2 Filtering out insignificant teams

There are a few criteria used within the original paper which exclude some of the teams from the analysis. There are two constraints which are applied for all analyses. First, we only consider repositories with at least 1 stargazer. Second, teams must have at least two updates per month on average within the timeframe. We take this second constraint to mean that there are at least $2T$ pushes total where T is the number of months of data within consideration.

The original paper found that there were 151,542 teams within the dataset after applying these filters. We found 150,039. We believe that these inconsistencies are due to the ambiguity in aggregating the raw data, as discussed above.

1.3 Preprocessing and calculating new statistics and measures

In order to actually create the figures, there are several measures that must be computed on top of what is provided from the aggregated data. The size of each team, the amount of work each team member does (by pushes, proportion of the team's work, and rank within their team), the effective team size, team experience, team diversity, and the number of leads on each team must be computed to create the figures. Some of these calculations (experience, diversity, and number of leads) are intensive as they involve iterating over all of the data several times. Currently, this step takes around one hour to run, and uses approximately 12 GB of memory to do so. We have already parallelized the computations of experience, diversity, and leads as they were very intensive. When we scale up to the full dataset, we will likely need to adapt this script so that we do not run out of memory. We will also likely need to enhance its speed as some of these calculations scale at least quadratically with the data.

We create two additional CSVs which are used to generate all of the figures:

- `repository_stats_expanded.csv`: Contains all measures from `repository_stats.csv` as well as total work of the team, effective team size, experience, diversity, and number of leads
- `work_per_member.csv`: Contains repo_id, username, amount of work, total work of the team, fraction of the work done by this user, rank of work for this user, and team size for all users and repositories.

1.4 Creating figures

We have reproduce all four figures from the original work. Almost all of the plots have slightly different values than the figures from the original paper, but we believe that this is due to the differences in aggregating the data which were discussed above. We believe that once these inconsistencies are ironed out, that the same code can be used, without modification, to reproduce the results of the original paper exactly. Original figures and reproductions are in `article.pdf`.

2 Burstiness and Memory Coefficient

The burstiness and memory coefficient analysis is split into three files:

- `burstiness_preprocessing.py`: calculate all τ's for all repositories and write to file
- `calc_burstiness_stats.R`: fit distributions to each repository and calculate burstiness and memory coefficient for each
- `burstiness_analysis.R`: analyze data, produce figures, calculate statistics, etc.

The process of computing and analyzing the burstiness and memory coefficient was quite challenging for two reasons:

1. I could not find implementations of Weibull and log-normal distributions with common parameterizations in Python. All of Scipy's implementations used standardized forms which did not seem to fit the model properly.

2. After resorting to R for fitting models and calculating statistics, I had to fight quite a bit with R to get things working quickly and smoothly.

To focus on my second roadblock, I had not previously used R to do anything computationally intensive, only simple plotting and statistical methods for statistics classes and some personal projects. However, running a somewhat expensive operation hundreds of thousands of times proved to take quite a bit of time to get running smoothly. My first pitfall was not preallocating lists. I am spoiled by python and it is efficient enough that most of the time preallocating lists is not necessary. This is not the case with R, however. I am also much less comfortable with data structures in R than I am in Python, so simply passing data structures to functions and so forth took much more effort than it normally would in Python.

Regarding my choice of analyses with these two measures, I had three objectives: choose the right model, analyze the joint distribution of B and M, and analyze the relationship of B and M with S. I based many of my figures off of what I had seen done in other papers that I read, specifically the Goh paper gave me lots of inspiration.

3 Scope of Project

The scope of the project decreased over time. I originally thought that reproducing the work of Klug and Bagrow would be quite easy, but it proved to not be. It took much longer and much more effort than I originally planned for. The next step was to then implement the burstiness parameter and memory coefficient. This also took much longer than I had hoped. I thought that implementing the fairly simple measure would be a streamlined process and the analysis would take up most of my time during this part of the project, but this proved to not be the case. About half of the time spent on implementing the two parameters was spent trying to decode what the original paper by Goh was actually doing. Meeting with Professor Bagrow helped some, but it proved to be much more challenging than I had hoped. The analysis of the burstiness and memory coefficient was fairly straightforward once I got down to it, but it did take some time. This is where I ended the project.

Every step took the time and effort that I originally thought the entire project would take. It was incredibly discouraging as my goals slipped further and further away even though I was putting in lots of work. It really puts into perspective the effort it takes to do even simple analyses.

I originally planned to also run analyses on newer data. This was the purpose of reproducing the work of Klug and Bagrow. However, due to time constraints I was unable to accomplish this. I am still proud of what I did accomplish, and incorporating more data will need to be a future project.