REPORT FOR SENIOR FIELD PROJECT

Xin Yao, Xi Qian, Giang Nguyen
TEAM GC, Brandeis University, Spring 2016

Common Crawl Report - Xin Yao

Common Crawl datasets have three different formats: WARC, WAT, WET, which contains the copy of web pages. WARC format is the raw data from the crawl, which not only stores the HTTP response from the websites it contacts (WARC-TYPE: response), but also stores the information was requested (WARC-Type: request) and metadata on the crawl process itself (WARC-Type). WET files contains only extracted plaintext. WAT files contains important metadata about the records stored in the WARC format above. Most importantly, the information of WAT files is stored as JSON which should be really easy to get data and process. That was why we originally used WAT files.

Common crawl provided repository that contains both wrappers for processing WARC files in Hadoop MapReduce jobs and also Hadoop examples to get you started. And we also wrote bunch of Java code which helped me to process the JSON file (WAT files) and got useful data from that. However, a technical problem was that the field of the target data on JSON file can't be accessed. For example, if a WAT file looks like this:

```
"A":{
        "B": {}, "C": {}, "D": { "E": {"data"} }
}
```

And we need "data" from "E". When tried to get accessed "E" block, we got an error saying that there was no "E" block. It was really weird because when printed each WAT file and we saw "E" block was just there. And then I tried to access data from other fields like "B", "C", and they all worked well except for "E". This problem was probably caused by some non-identified characters. Finally, we changed our approach and used regular expression.

Moreover, common crawl datasets are super big. Averagely, the datasets for each month is more than 100 Terabytes. For example, "WAT.file" for April 2014 contains 44488 dataset URLs. Each URL is connected to an online dataset (each dataset has many web pages in WAT format.) which is averagely 4 GB or more. Sometimes, when you try to contact with dataset URLs, it is possible to get some exceptions raised. (Sometimes s3service terminated, or data URL has been closed, or my network speed was not fast enough etc.) Based on these situations, we random picked 0.1%

of data for each month (read every 1000 URL in WARC/WAT files), which still took about 2 hours to process. So for the both 2014 and 2015, we need at least 36 hours to process all the datasets. Unfortunately, once any unexpected data appeared in the database, we would need to fix regular expressions and re-process the data of that month again.

After several team meetings, we decided to extract web URLs and IP addresses from WAT files, and to extract technical related information in HTML tags from WARC file. Based on counting the number of different web URL in certain month, we can roughly estimate which website has been largely visited in that month, and IP address of that website can tell where that website server(s) is/are located. Some advertisers may need these information to see which website is worth the investment. We also believe that some built-in technologies for web pages can be read from HTML tags, so we spent a lot of time to understand how web pages use different technologies and recorded their formats to build different regular expressions for each technology. However, using too many different regular expressions in the program will definitely reduce the working efficiency and increase the time of processing.

Our final datasets can be viewed as two main parts: the first part is about web information, which tells the number of different web URLs that has been visited by crawler in particular month, and what kind of technologies has been built into each website; the second part is about technology information (we only focused on jQuery, Ionic, Hubspot, Angular, and React such 5 categories). I extracted these technologies from crawled web pages, so the number of each technology appeared in particular month is absolutely influenced by the website that technology belongs to. For example, if website A contains technology B one time, and website A is visited by crawler 10 times a month, then the number of technology B in that month will be 10 too. So, based on my datasets, we can roughly see that which website is the most active monthly, and what technologies it uses, and how those technologies appear monthly.

If we had more time, we would try to use the new index and query API system of common crawl. This new feature was made by common crawl community in January 2015, which provides an API that allows user to query a particular index for a particular domain, (index is a snapshot for each crawling run they perform), and it will return back results that point you to the location of where the actual HTML content for that snapshot lives. Moreover probably do some semantic analysis with WET files would also be helpful.

Report on Reddit / HackerNews - Giang Nguyen

Our original idea when choosing these two sites as data sources was to acquire data regarding technologies mentions in social networks context. Reddit is an entertainment, social news

network where users can submit as text posts or direct links, essentially making it an online bulletin board system. Redditors can downvote or upvote a comment/post and the total score of that is assigned to the original poster of that comment/post as “karma” points. Hacker News functions very similarly to Reddit with the exception that it puts a heavy focus on computer science and entrepreneurship. Hacker News users cannot downvote posts but they can do so with comments. It also employs the “karma” point system.

As we work on acquiring data from these two sites, it gradually became clear that it would require a lot of work to achieve what we wanted at first. Reddit has an API limit of 60 calls per minute. Every single comment / post takes an API call, effectively making it impossible to get all the content (a front page post may have thousands of comments and there are hundreds of them every 24 hours). Even if we only want the content of the top posts, there was not an efficient way to pinpoint an exact date to get all comments / posts of that moment. The reason behind this is Reddit changes constantly: every thing can be downvoted/upvoted at any given time even if the content is years old. For Hacker News, the current API living on the firebase server apparently has no limit. However, it has the same problem as Reddit. An API call can only get one item and there are at least 12 millions of them. With more time, we believe all items can be scanned though.

Already digging deep into these two sites, we decided not to diverge our efforts re-learning other data sources but rather made the best use of what we had. We put focus on the social aspects of the owners and users who created and used the GitHub repos. We found out that it was pretty likely for a GitHub user to have the same handle for his / her Git account and for Reddit / Hacker News. A Python script made using the Python Reddit API Wrapper (https://github.com/praw-dev/praw) allows us to get a user’s total karma and break it down by subreddits (a subreddit is a smaller reddit dedicated to a community who share an interest, e.g. soccer, coding, religious belief…). Although we did not use the breakdown feature in our final scoring function, we believe it could be very useful for future references when learning about which communities the developer associates with. A very similar script was made for Hacker News karma point of a particular user.

Another idea became relevant in the later part of the project was that whether we could track developers’ activities based on their emails. Some users do have their emails publicly available on GitHub, however most don’t. We had an idea of surfing through LinkedIn API because many developers list their GitHub accounts or at least projects’ names on their LinkedIn profile page. The issue with LinkedIn was that an authentication from a LinkedIn account had to be made for every API call and that would make searching automation not very viable. Besides, it was not easy to decide when we would cross the lines of privacy if in fact we could use automated search to track real life activities of developers.

For possible future continuation of this project, we think it would be better to use Reddit / Hacker News to track a particular interest and use other data resources to get the big picture. Stackoverflow, The Internet Archive and other social media services like Facebook, Twitter, Vk, Weibo seem to have a lot of potential for that. Also, we suggest using our methodology but in a much larger scale, handling big data to get all the scores possible and compare them with each other. If possible, we would try to draw correlations and heatmaps of developers' associated communities.

Report on GitHub - Xi Qian

GitHub came into the world in 2008 and has more than 14 million users and more than 35 million repos as of April 2016. It has been the most popular open source hosting service in the world. Famous projects on GitHub includes Bootstrap, django, rails, redis, swift, etc. There has been many products exploring the data from GitHub and there has been anual GitHub Data Challenge where competitors raced to build interesting analyzing, mining and visualization using GitHub. GitHut (http://githut.info/) is one of them. GitHut visualized the trend of each programming language's growing path from 2012 to 2014.

Our initial approach was trying to utilize the same data source as GitHut, the GitHub Archive and Google BigQuery. GitHub Archive (https://www.githubarchive.org/) records the public GitHub timeline and made the datadump avaible for easier access. Google BigQuery made the task even easier. The data are pre-loaded into a database and users can issue SQL queries to retrieve desired results. However, there were a major API change for GitHub at the end of 2014, data of the most organized format on either GitHub Archive or BigQuery were discontinued. Later data were harder to process.

We came to the point where we decided to record our own data with calling GitHub API directly. GitHub API's rate limit for authenticated users is 5000 calls per hour, and for unauthenticated users is 60 calls per hour. We attempted to store the returned json from an API call into individual json files. With more than 35 million repos, the expected size of the metadata returned can be about 130 GB. And based on the average response speed (about 1 call/second), it could take more than 4 days to download them all, if GitHub hadn't blocked our ip address. With this in mind, we used the first 30,000 repos for our next step of exploration, and downloaded 10 GB of data as preparation for scaling up.

With the data, or at least a sample of it, we started looking for ways to evaluate repos. Our first thought was the language. We wrote a script count the lines and files of each language. Although

it would very long time to iterate through all repos, a pilot run on the first 1,000 repos showed Ruby being the most commonly used language. Which was understandable because GitHub was written in Ruby and those repos were created at the very first stage of GitHub.

Another attempt was trying to evaluate the users, and in turn evaluate the repo according to the users that contributed to it. It's very easy to get the owner of a repo, but to get the list of contributors and amount and time of their contribution, it would require going through the event history of the repos, which we cannot possibly do to every repo and find the best within reasonable time.

There has also been an episode of whether to dump the returned json's into a relational database. The convenience of having a database and using SQL queries in the future is tempting, but we later realized it's extra work to analyze the structure of the json's and handle different cases when the json won't fit into our expected schema.

To produce a working product, we finally narrowed our vision to the commit, fork, and issue event history. Commits represent the owner's efforts on the project, forks represent the community's interest and urgency in optimizing the project, and issue events represents the interaction between the owner and the community, and may be extended to common users instead of limiting to coders. The details of scoring a repo will be described in the next section.

As suggestions for the future, we can consider more about how to incorporate user's value into the evaluation of repos. With the history we can find the user connected with each event, and there are repos, other users, and events connect to that user. With some iterations, the evaluation of users and repos may come to a fixed point and that probably will be score we were looking for.

More generally, I would say simply with the GitHub data source alone, it would be worthy of a team's investment. With more manpower, we can dig deeper into the information behind the tons' of data. Although we did not follow through our attempt with GitHub Archive and Google BigQuery, these still are great data source and still worth look into. As time changes there will be more and more repos on GitHub. New tools and new information may come into light at any time.

Scoring Report - Giang Nguyen

We came up with the idea of a hotness score to assign for every technology, website and GitHub repo we found. However it turned out scoring technologies and websites are not very simple.

When a website and its component technologies and services are discovered, we still could not figure out how to assign the scoring weight to each of those. The ultimate question was whether the website made these technologies hot or the other way around. In the end, we decided to present them as graphs showing the trend rather than going through with scoring them.

Yet, we found out a relatively smart way to score the open source projects on GitHub. We have the preliminary score which shows first when you search, which is just stars and forks of the repo combined. This pre-score lets us have a quick glance at the repo and once we know which repo we want to see further, the final score is generated. The final score formula is as follows:
(Commit_Score + Fork_Score + Issue_Score) * (Reddit_Score + HackerNews_Score)
When we run the GitHub script we have the three history files for commits, forks and issue events. From that we would generate the first three subscores of the formula. A history file tracks the activity of the corresponding type by month. Since we are more interested in the recent technologies, we assign higher weights to more recent months. This results in a geometric weight system that for every month backwards in the past, the value of that month is changed by a multiplier which degenerates monthly as well. We use a 90% multiplier coefficient so the most recent month would be 100% of value, the next ones would be 90%, 81%, 72.9%, … of value. For the Reddit and Hacker News score, we would take the log base 10 of each total karma score of a user from those two sites and then add them together. If the karma score is less than 10 (in case a user does not have an account there), we assign a value of 1 to that score and continue calculating as usual.