Exploratory Analysis of Github Archive

Student 1: Valtteri Vuorio

Student 2: Svetlana Pelevina

Project source code

1. PROJECT DESCRIPTION

The purpose of the research project is to get familiar with Spark analytics engine and gain experience in Big Data analysis. During this project, we will conduct an experimental evaluation of data obtained from the GitHub Archive Project. We visualize and analyze data obtained from their API using methods, algorithms and techniques available in the Spark analytics engine. Examples of expected results are analyzing the sentiment of GitHub commit messages, creating statistics for frequently used languages, and popular companies. The expertise needed for this project is basic knowledge of statistics, machine learning and big data methodologies.

2. DATA DESCRIPTION

2.1 Github Archive

GitHub Archive [1] is an open-source dataset, which includes logging and archiving GitHub's public event data. The dataset is continuously updated, providing data for every hour since 2015-01-01 to yesterday. Each archive contains JSON encoded public events which happened on GitHub during a given hour. GitHub provides 20+ event types, which range from new commits and fork events, to opening new tickets, commenting, and adding members to a project . These archives are quite huge: for example it was 500 GB for 2015, 1.07 TB for 2018, and a single hour of events from yesterday (15.04.2021) was roughly 600MB. The JSON archives can be accessed either by visiting the link with a web browser or by using common python modules for HTTP requests and file extraction.

The basic unit of data in each JSON file is an event that describes some user behaviour, for example pushing code or sending a comment. Note that each event contains around 900 fields, most of which have no use in this project - this imposes some challenges when storing the data for later use. To demonstrate the different types of events and their relative occurence we summarize a single hour of activity from yesterday (15.04.2021) in Table 1. There were a total of 154260 events, the large majority being push events.

Event	*Count*	*Event*	*Count*
PullRequestReviewEvent	5500	IssueCommentEvent	9316
PushEvent	81608	DeleteEvent	4926
GollumEvent	494	IssuesEvent	3955
ReleaseEvent	952	ForkEvent	2779
CommitCommentEvent	636	MemberEvent	612
CreateEvent	20019	WatchEvent	8103
PullRequestReviewCommentEvent	3661	PullRequestEvent	11248

Table 1. Single hour of activity from yesterday (15.04.2021).

The information that we are interested in this project are comments, the related programming language and the owner of the repository. We found that these three are only present in pull requests, pull request reviews and pull request comments, which will be the source of data in this project. The figure below shows the structure of the event with the fields that we consider in our analysis.

Event structure	**Data example**
root	
|-- **created_at**: string *(nullable = true)*	*2021-04-29T13:59:59Z*
|-- **payload**: struct *(nullable = true)*	
| |-- **comment**: struct *(nullable = true)*	
| | |-- **body**: string *(nullable = true)*	*Oh good catch*
| | |-- **created_at**: string *(nullable = true)*	*2021-04-29T13:59:59Z*
| |-- **pull_request**: struct *(nullable = true)*	
| | | |-- **repo**: struct *(nullable = true)*	
| | | | |-- **forks_count**: long *(nullable = true)*	*123*
| | | | |-- **language**: string *(nullable = true)*	*Python*
| | | | |-- **open_issues_count**: long *(nullable = true)*	*10*
| | | | |-- **stargazers_count**: long *(nullable = true)*	*56*
|-- **public**: boolean *(nullable = true)*	*true*
|-- **repo**: struct *(nullable = true)*	
| |-- **name**: string *(nullable = true)*	*/google/forms*
|-- **type**: string *(nullable = true)*	*PushEvent*

Figure 1. Overview of an event

2.2 The Sentiment140 Dataset

Another dataset we considered was a corpus of labeled sentiments for the purpose of sentiment classification across Github communities and programming languages. While there exists a number of sentiment datasets, there is still a general lack of large-scale datasets due to the labor cost of manual labeling. We chose the Sentiment140 [2] dataset because of it's accessibility and large volume (1.6 million records). Each record is a Tweet (i.e. short sentence) and a labeled sentiment (negative or positive). The limitation of this dataset is that it does not contain neutral sentiments and it is questionable that a model trained with this dataset produces reliable results in other domains. However, the discussion at Twitter is quite broad and covers a wide range of topics.

Sentiment	*Username*	*Tweet*
Positive	Lauren2206	woooo summer sun!
Negative	mybirch	Need a hug

Table 2. Example data from Sentiment140 dataset

3. METHODS AND TOOLS

3.1 Python ecosystem

We use the Python ecosystem for basic tasks such as requesting data from the API endpoint , extracting archives, and managing files. Numpy and Pandas are used for some of the data processing and Matplotlib is used for drawing figures.

3.2 Spark Core

According to the project requirements and course content we use Spark to accelerate our data processing and analysis. The Spark Core contains the functionality for distributing data processing to each worker in our cluster, which contains a total of 4 workers, 8 cores and 12gb memory.

3.3 Spark SQL

Spark SQL is a module of the Spark ecosystem that contains functionality for working with structured data. This is especially useful for processing Github events, as the raw data is unstructured. Spark SQL will automatically structure the data for us and let us execute advanced data processing and SQL queries over the structured format.

3.4 Spark MLlib

Spark Mllib is another module of the Spark ecosystem that contains functionality for machine learning tasks. We intend to leverage Spark MLlib for performing sentiment classification using the Sentiment140 dataset.

4. DATA ANALYSIS

4.1 Research questions

The main focus of our research is to analyze the sentiment across programming languages, Github communities and different time periods. The results should give insight to a number of interesting research questions:

Sentiment across the most popular languages: There is a strong stigma amongst developers that certain languages are more pleasant (Python?) and some of them are frustrating (Java?). Which languages are used by the happiest and most frustrated developers?

Sentiment across companies and repositories: There is quite an interesting discussion about the relation between sentiment and company size. We're going to get the size of the company based on the number of forks, and also look at well-known companies like Google, Amazon and Facebook. Which companies and repositories have the happiest or frustrated communities?

Sentiments across different time periods: How has the general sentiment changed over time for above mentioned companies? Is the day of week or the hour of day correlated with average sentiment?

4.2 Data pipeline

The first obstacles in this project was how to deal with the large volume of data acquired from the Github Archive. As mentioned, a single hour of raw data is roughly 600MB, so we quickly run into memory limitation problems when processing data over a timespan of several hours, days or months. Given the limitation of ~5GB in our cluster we adapted the data pipeline in Figure 3. The data cleaning process removes unnecessary data and refines it, such that we can store intermediate results in memory for later use. After processing data from a longer timespan we collect the intermediate results and conduct long-term analysis and visualization.

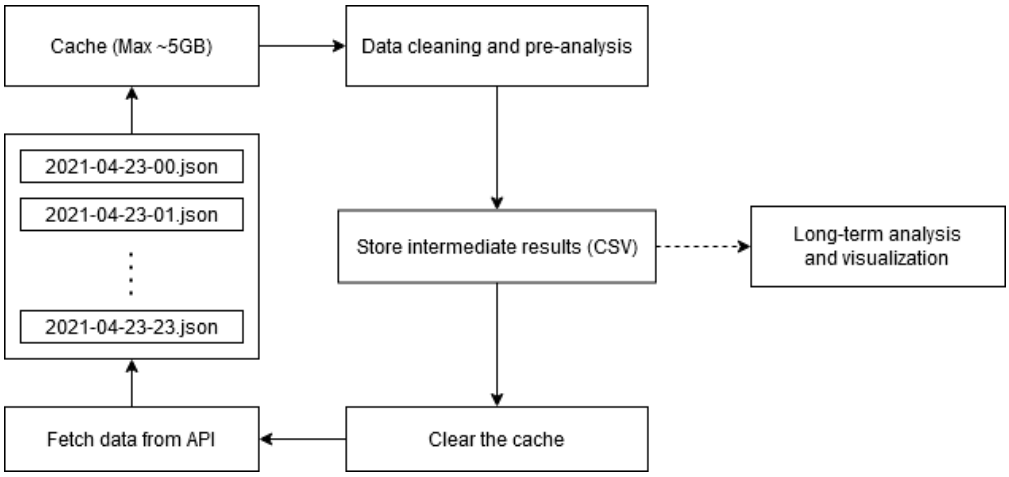

Figure 3. The data acquisition pipeline

To conduct the long-term analysis, we retrieved data spanning several days and stored intermediate results to a data structure shown in Figure 4. This data was small enough to store in a single file in our cluster server. For example, the language analysis file is XX MB and contains XX data points. The data is stored in CSV files, as this is a fairly simple format for big data, and it is convenient to work with.

```
root
 |-- Repo: string (nullable = true)
 |-- Language: string (nullable = true)
 |-- Comment: string (nullable = true)
 |-- Forks_Count: integer (nullable = true)
 |-- Stargazers_count: integer (nullable = true)
 |-- Open_issues_count: integer (nullable = true)
 |-- Date: timestamp (nullable = true)
```

Figure 4. The structure of the intermediate data used for long-term analysis.

4.3 Sentiment classification

To explore the sentiment across Github communities, we used Spark Mllib to train a model that predicts sentiment from a given sentence. The aim was to choose a relatively simple model that we can implement within the time constraints of this project. An overview of the pipeline is shown in the below figure.

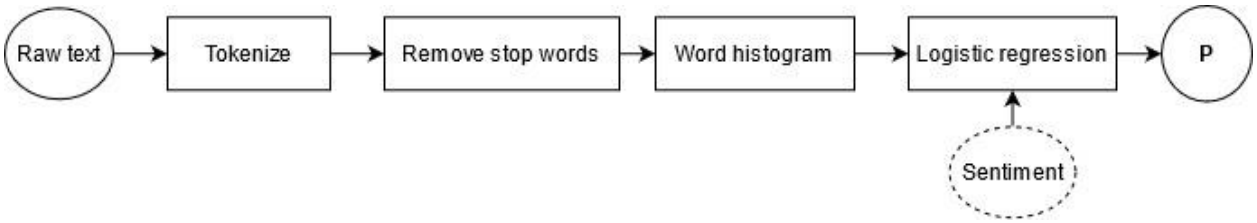

Figure 5. Our sentiment classification pipeline.

The details for each step in Figure 4 are explained below.

Tokenize: We use the regex "@\\S*|\\W|http\\S*" to skip usernames (e.g. @Lauren2206) and links (e.g. http://google.com).

Remove stop words: We remove a predefined list of total 181 common stop words from the sentence. These include words such as “be”, “you”, “then”, etc.

Word histogram: The word histogram is constructed by counting the frequency of each word. We limit the vocabulary to 100-100000 most frequent words and set the minimum frequency to 5.

Logistic regression:

Finally, we use logistic regression to predict the given sentiment label (positive, or negative). In this model, the probability of a positive sentiment (p) is predicted as a linear combination of word histogram frequencies (**x**) applied to the logistic function:

$$p = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

(Equation 1.)

The implementation in Spark MLlib [3] [4] optimizes the weights of the model (**w**) using regularized logistic loss. The logistic loss is a special case of cross entropy applied to a binary response variable and the regularization is defined as the absolute sum of all weights. The loss function (L_total) is minimized using stochastic gradient descent:

$$L_{\text{total}} = \lambda ||\mathbf{w}||_1 + \frac{1}{n} \sum_{i=1}^{n} L(\mathbf{x_i}, \mathbf{w}, y_i)$$
$$L(\mathbf{x_i}, \mathbf{w}, y_i) = \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x_i})), y \in \{-1, 1\}$$

(Equation 2.)

To validate and improve our model, we partitioned the Sentiment140 dataset to training and testing sets at a ratio of 90% / 10% and then conducted a hyperparameter search on two parameters: The regularization parameter (lambda) and the vocabulary size of the word histogram. We measured the F1-score against the test split and chose the parameters resulting in the highest score. The results are seen in Table 3 below.

Reg. / Vocab size	100	1000	10000	100000
0.000	0.645073	0.744660	**0.777977**	0.776525
0.125	0.644638	0.743337	0.775839	0.777695
0.250	0.643508	0.742042	0.774576	0.777197
0.500	0.643352	0.740815	0.772755	0.775840

Table 3. The F1-scores for each combination of regularization parameter and vocabulary size.

The final F1-score, 0.78, shows that the model can indeed perform sentiment classification, but the score is relatively weak and it is expected that it will fail consistently. However, we can still conduct meaningful

comparison with the model given that we are interested in the average sentiment of a large collection of sentences, for example under a single repository.

In Table 4, we can see some sample sentences and the model predictions. The symbol P represents the probability of the text having a positive sentiment. In reverse, it is the inverse probability of the text having a negative sentiment.

p	*Sample Text*
0.06	Your code sucks and you will never get a job.
0.94	I love your library, very excellent code sir.
0.11	Why did you choose Java? It is so f***ing slow and terrible.
0.92	Yes, thank you very much friend.
0.47	I should be around 0.5.

Table 4. Sample sentences and model predictions

4.4 Analysis description

We analyze the GitHib archive data in 3 different ways:

Languages analysis: We look at all language sentiments and analyze them by sorting it. We also look at the most popular languages by the number of forks and stargazers in the repository and compare moods in terms of time (for April 2020 and 2021).

Companies and repositories analysis: We look at the most popular repositories and the most popular companies, where the company name is determined by the repository prefix. We assume that the most popular companies, such as Facebook, Amazon, and Google, should be interesting to look at. Then we compare their average sentiment value in terms of time.

Time analysis: Finally, we look at the average value of **p** over different time spans, including the day of week and hour of day.

Using SQLContext DataFrame Operations groupby() and agg(), we group the data by criterias described below and calculate the average value of **p**, as in Equation 1, for each group. The simplest interpretation for this value is a measure of sentiment between 0 and 1, which will be used for comparison in the results section. In addition to this we use a categorical mapping where positive (p >= 0.8), neutral (p from 0.4 to 0.8) and negative (p < 0.4) to measure the total volume of positive, negative and neutral comments.

In the next section, we will show the results of the described analysis and show the graphs created using Pandas and MatplotLib.

5. RESULTS AND DISCUSSION

5.1 Language analysis

We extracted data for the period from 01.04 to 31.04, calculated the results as explained in section 4.4. You can see the result below (Fig. 6).

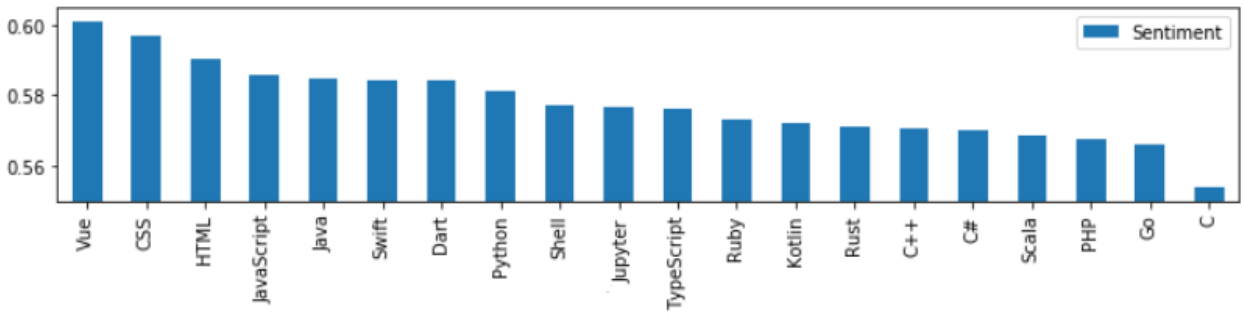

Figure 6. The 20 most frequent languages sorted by sentiment (April 2021).

Interestingly, developers using the web development stack (CSS, HTML, Javascript) score the highest sentiment. The highest score goes to Vue, which is a Javascript framework for web development. Contrary to popular belief, Java developers seem to be relatively positive. On the other hand, Scala which is another JVM language is at the bottom of the list. The lowest score at this list is C, a language that is known to cause frustration amongst developers. Overall, the C-language family (C, C++, C#) turns out to have relatively low scores.

Let's look at the moods of the most popular languages for the period from 01.04 to 31.04 for different years. Let's assume that the popularity of the language is related to the number of forks and the number of stargazers in the repositories. You can see the result below (Figure 7). The size of the bubble indicates the average number of pull requests for these languages.

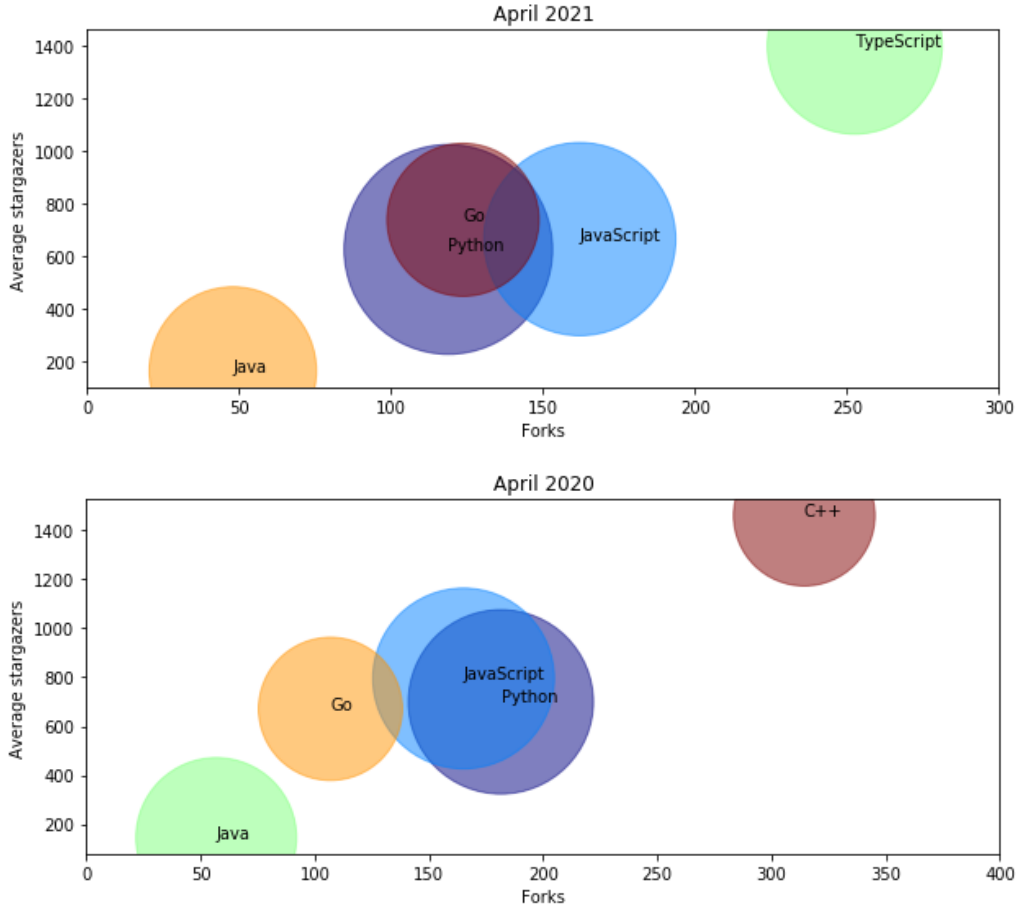

Figure 7. Most popular languages in April 2020 and 2021.

As we can see, the most popular languages in both years are Python, Java, JavaScript, and Go. Let's compare the moods of the first 3 of them. We calculated the average sentiment value for each language and for each sentiment group in April 2020 and 2021. You can see the result of the calculation below (Fig. 8).

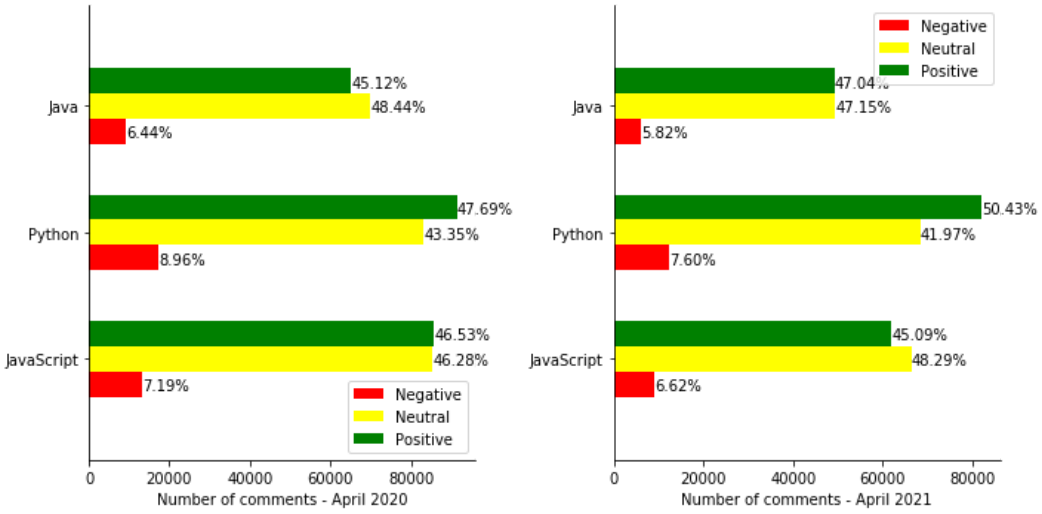

Figure 8. Sentiment of the most popular languages in April 2020 and 2021.

These 3 languages are completely different: Java is an OOP language and quite complex to start with, Python is usually assumed to be an easy-to-use language and which is quite good to start with, and JavaScript is a web language that has quite a few strange features, especially in mathematical operations. Therefore, we expected to see completely different results for these languages. But eventually, the sentiment towards these languages is almost the same in terms of the percentage in the groups, and the sentiment is mostly positive and neutral. However, Python has the highest percentage of positive comments, as well as for negative comments. But for all languages, negative comments decreased in 1 year, and positive comments increased, except for JavaScript.

5.2 Companies and repository analysis

Moreover we decide to look at the sentiment for the different repositories and companies.
We extracted data for the same period as in section 5.1 and calculated the results as explained in section 4.4. You can see the results below (Fig. 9).

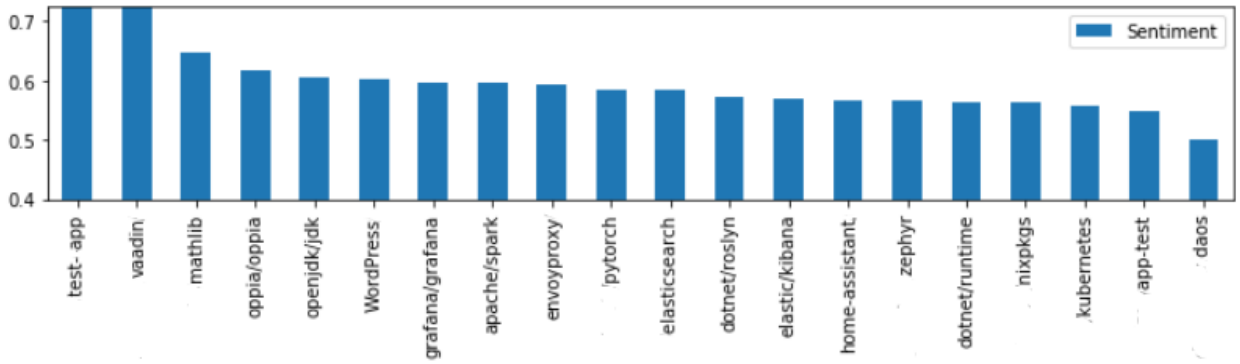

Figure 9. The 20 most active repositories sorted by sentiment (April 2021)

There are quite a few popular tools on this list, such as Matplotlib, OpenJDK, and Spark. Looking at the results more closely, they reveal some limitations of the study. “Test-app” is a repository that only contains automatized

pull request events. Looking at the “vaadin” repository, it also has a code review bot that seems to have a strong effect on the result.

Let's also look at the average sentiment of popular companies, we decided to analyze Google, Amazon, and Facebook as fairly well-known examples of this. We calculated the average sentiment value for each company and divided it into groups in the same way as described in section 4.4. The result is shown in Figure 10.

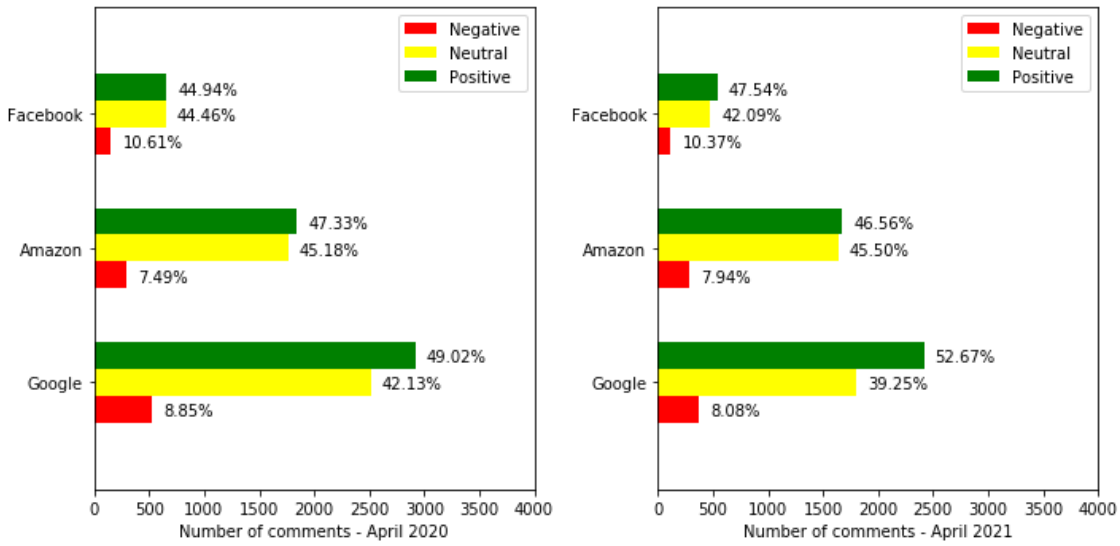

Figure 10. Popular companies sentiments in April 2020 and 2021

As for the previous section, the situation has not changed much over time and the results are not stunning. However, Google is the leader in positive comments, and Facebook is the leader in negative comments. For Google and Facebook, the sentiment of the comments got a little better, the positive increased, the negative decreased, but for Amazon, the situation is the opposite.

5.3 Time analysis

As stated in section 4.4, we investigated the average sentiment over different time spans. The following figure shows the average hourly sentiment between 01.04.2021 - 30.04.2021.

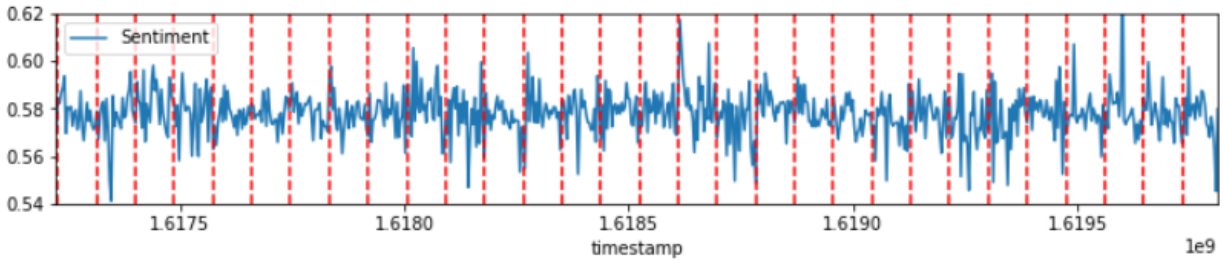

Figure 11. The average sentiment measured at each hour of the month.The start of each weekday is marked as a vertical line.

Looking at Figure 11 more closely, there seems to be a noticeable theme between different hours of day and day of week. To further investigate this we looked at the average for each hour of day and each day of week for the entire month. The results are shown in Figures 12 and 13.

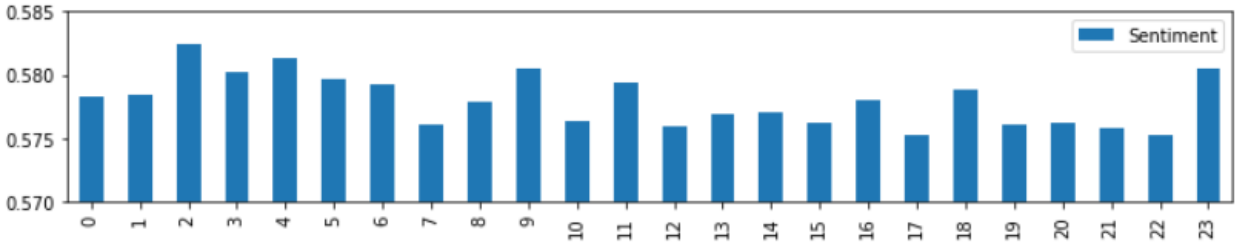

Figure 12. The average sentiment of each hour of day.

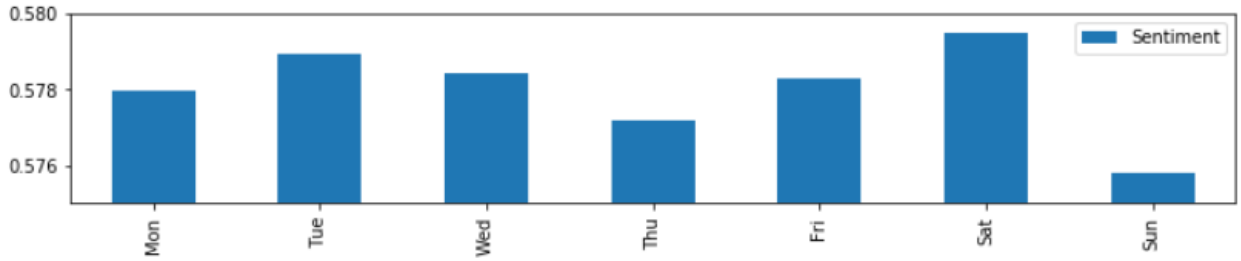

Figure 13. The average sentiment of each weekday.

According to the result, night-time developers tend to be slightly more positive than day-time developers. The most negative day of the week is Sunday, followed by Thursday. Developers tend to be happiest on Saturdays and Tuesdays.

5.4 Challenges and limitations

Data volume: The most difficult part of the project work was getting the data. Using the algorithm described in Data Analysis, retrieving and storing data for 1 month takes around 6 hours. Since sometimes some errors occurred during the extraction, this process takes a long time.

Time constraints: Another difficulty we encountered was the time limit. Since the project timeline is quite tight and we spent most of the time extracting the data, we were not able to add more dataset analysis.

Validity of the classification model: There are a number of issues that undermine the credibility of the classification model. First of all, the logistic regression is a relatively simple model that operates based on word frequencies alone. This could be further improved by adopting some state-of-the-art models instead, such as LSTM or Transformer. However our development environment did not contain the resources to implement these. Another issue is that we did not study in detail how well the Twitter dataset translates to our data sources.

Bots and non-English languages: In retrospective, we noticed that a large portion of the data was produced by bots that send automatized messages to repositories, which potentially distorted the results. Another issue is that Github communities are not always using the English language but our model was trained on English sentences.

5.5 Future work

GitHub Archive contains a huge amount of data and can be analysed in different ways. For example, it is possible to get GitHub user location using API and url in the Actor field. But as for our research, we can improve the following:

Data extraction algorithm: The existing data extraction algorithm is quite slow. It would be much more efficient if we could extract the data faster. Moreover, this algorithm is not perfect and can extract data with some defects. For example, some comments were received with “/n " characters, which result in invalid strings in CSV files.

Analysis of a longer time period: For the reason described in the previous paragraph, we can only get data for 2 months. It would be more informative to look at the data for the whole year. But since the data for 2018 is 1.07 TB, this is quite complicated and requires a good and efficient algorithm.

Taking full advantage of the Spark ecosystem: Spark is a powerful data analysis tool. We can take advantage of many more features that Spark can offer us. For example, we can analyze the correlation between different variables or predict future sentiment for a language or company.

CONTRIBUTIONS

Student 1: Valtteri Vuorio:

- Implementation of data fetching from API using http requests
- Retrieving data for April 2021 and applying the pipeline described in section 4.2
- Implementation of the classification pipeline described in section 4.3
- Using spark SQL to get results for language, repository and time analysis
- Using MatplotLib to create graphs for visualization
- Writing the report

Student 2: Svetlana Pelevina:

- Retrieving data for the April 2020 and April 2021
- Fix bugs in implementation
- Using spark SQL to get results for language, repository
- Using MatplotLib to create graphs for visualization
- Analysis of the results of the received data
- Writing the report

CONCLUSION

In conclusion, we described a method to leverage the Spark ecosystem for processing and analyzing data from Github Archive and presented our results. The focus of this study was to train a machine learning model for sentiment classification and use it to analyze various aspects of the Github community. The results show interesting insights on the mindset of developers within different programming languages, repositories, companies and within different times of day and day of week. Finally, we discussed some key limitations of the study and future research directions.

REFERENCES

[1] https://www.gharchive.org/
[2] http://help.sentiment140.com/
[3] https://spark.apache.org/docs/latest/mllib-linear-methods.html
[4] https://spark.apache.org/docs/latest/mllib-optimization.html