Analyzing Responses on Online Developer Community, StackOverflow

Class: CAPP 30123 / CMSC 12300 (Computer Science with Applications III)
Group name: HackyStacks
Group members: Adam Shelton, Dhruval Bhatt, Li Liu, Sanittawan Tan

Introduction

This write-up reports the outcome of a final project for CAPP 30123/CMSC 12300. The first section discusses the background of the project and the research hypothesis. The second section documents our data set and methods for preprocessing. The third section discusses our exploratory analysis. The fourth section details the main analysis which provides an answer to our research

hypothesis. The final section provides a high level summary of our project. The extrapolated runtimes and discussion of challenges are included in each section or task where applicable.

Background

This project was born out of our curiosity about the fact that in recent years Stack Overflow has been criticized by various users on and off social media for being unwelcoming to new users. In fact, Stack Overflow also recognizes that this is a serious problem since they are losing old contributors and the criticism and widespread negative attitudes on the website shun new users. Because the website makes its data available online, we were able to form hypotheses and analyze the issue. We suspect that the hostile attitudes towards new users could be correlated with the maturity of scripting and programming languages or frameworks.

Hypotheses

We question if answer providers on Stack Overflow become more impatient and meaner as scripting and programming languages or frameworks become more mature and popular. However, we need a reliable measurement of "impatience" or "hostile attitudes." Thus, we refined our question as follows: as programming languages mature, have sentiments of the answers become more negative? We hypothesize that as scripting and programming languages or frameworks become more mature and popular, the sentiments of Stack Overflow answer providers become more negative.

H: Stack Overflow answer providers become more negative when answering questions as scripting and programming languages or frameworks become more popular or mature.

Testing

To test our hypothesis, we employ the following procedures:

1. We processed the data set so that it is in the form that is appropriate for our method of analysis.
2. We did an exploratory analysis on the top 15 scripting and programming languages or frameworks based on their popularity
3. We then cross-checked the list with the scripting and programming languages or frameworks that are relevant to the class.
4. We conducted other relevant exploratory analysis to gain more understanding about users, questions, and answer providers.
5. We conducted the main analysis.

Data

Data Description

The data set was originally downloaded in bundle from Archive.org.[1] The table below details the breakdown of our data set. The data set comprises of 8 sub files. We only utilized the ones that are highlighted in red.

No.	File name	Type	Size	Lines	CSV File Size
1	Tags	XML	4.7 MB	54,467	5.6 MB
2	Post Links	XML	667 MB	5,696,052	NA
3	Users	XML	3.1 GB	10,097,980	1.79 GB
4	Badges	XML	3.4 GB	30,347,227	3.28 GB
5	Votes	XML	16 GB	167,002,408	NA
6	Comments	XML	18 GB	71,968,802	NA
7	Posts	XML	66 GB	43,872,994	22.5 GB
8	Post History	XML	113 GB	< inf[2]	NA

Table 1: Describing data size

Data Preparation

Most of the **cleaning and data processing** was done using MPI in Python. While many of the analyses relied on combining and condensing data from Stack Overflow into a new dataset of different measurements, for which MRjob was well suited for, this was not the case for the file processing and cleaning. In these instances, each line of a file needed to be processed, but as each line could be processed independently of each other, this made MPI uniquely suited to handle these situations. As such, we used MPI to handle two main tasks, converting the raw data from the Stack Overflow data dump into a format that could be more easily used in the analyses, and for converting the output files from these analyses into a format that could be more easily used for summarizing our results.

The raw data from the StackOverflow data dump came in XML format, which was difficult to work with in its initial form. XML files are hard to interpret on their own, as columns cannot be separated just by splitting each line by one character, like a CSV. Therefore, we wanted to use a pre-built parser to get the text data from the XML file and write it to a CSV file. For the XML parsing, we used xml.sax, as it could process files line-by-line and allowed for easy access to the keys and their values for each XML tag. As the values were not in the same order for each tag, knowing what key it corresponded to was important to putting that value in the right CSV column.

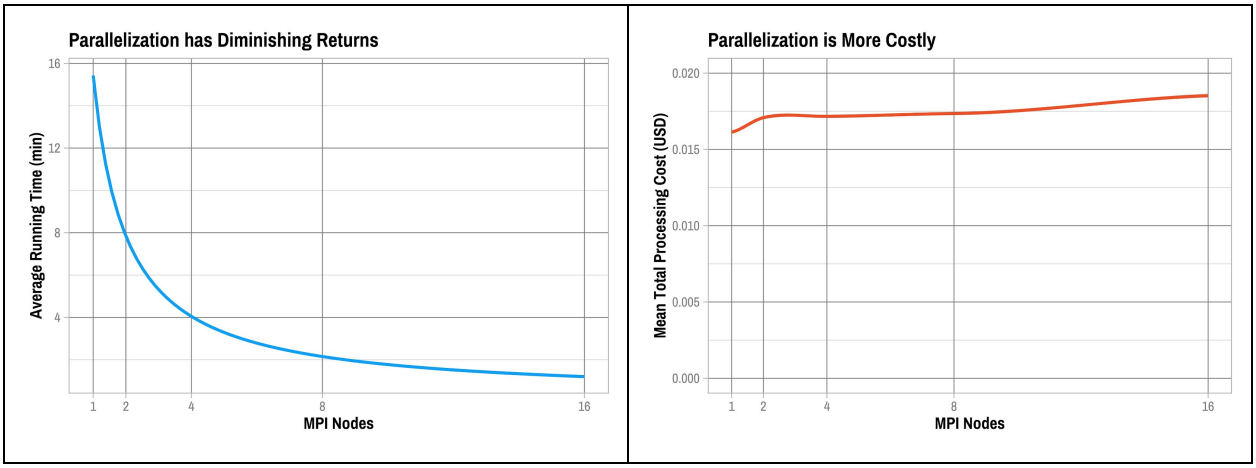
However, the xml.sax parser would not process a line of XML on its own, likely due to the hierarchical nature of XML files, so each file needed to be split into several files to parallelize the process

[1] https://archive.org/details/stackexchange

[2] We have some trouble counting the total number of lines in this dataset. It failed with broken pipe error multiple times and we could locate the source of this error.

using MPI. Once these files were split, the correct XML header and root tags needed to be added to the split files, so the parser could read them correctly. We could have accomplished this using Python code but decided that the methods for doing so would have been likely more inefficient than the built-in command line tools in Linux. Therefore, we executed those tools from the Python script to allow everything to work seamlessly, while maintaining the best efficiency in our code.

Using this code, we ran some tests to see which MPI configurations gave us the best performance on Google Cloud. To do this we ran 10 iterations of each configuration, to help control for any external factors. Five configurations were run on a single host (VM), with one, two, four, eight, and sixteen MPI nodes, while two additional sixteen node configurations were run on two and four hosts. This allowed to gauge performance over increasing numbers of nodes and hosts. For each of the hosts on Google Cloud at least double the amount of cores were provided as MPI nodes for that host. During these tests we measured only the time it took for just the conversion process to complete, after all splitting and preparation of the XML files was completed on each host. The mean time for one MPI node on one host was 918 seconds (a little over fifteen minutes), while for sixteen nodes on one host was 66 seconds, compromising an improvement by a factor of about fourteen. Sixteen nodes across two nodes had a mean processing of time 72 seconds, while those same sixteen nodes across four hosts completed the task in a mean of 64 seconds. While it makes sense that more MPI nodes creates a proportional increase in performance which is associated with a slight increase in cost, it is unclear why multiple hosts appears to affect performance in this manner, however the differences in those results are within the margin of error.



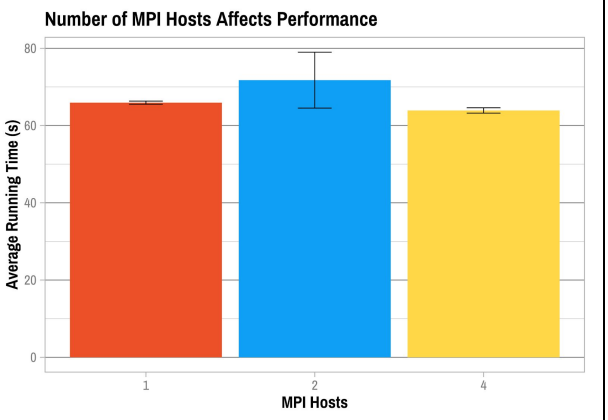


Figure 1: MPI Runtime analysis

The second task for which MPI was used is to **convert the text file outputs** of the other analyses into CSV files that could be easily used for the tables and visualizations needed to showcase our results. As these text files typically contained some combination of strings and Python lists as printed strings, we used the literal_eval function in the ast package to parse the string representations of these objects into actual objects, whose elements could then be written to a CSV file. In contrast to the previous MPI script, this one used traditional scatter and gather methods to distribute data among the different nodes. While we thought these output files would be small enough to be distributed in this manner, there were still issues with the size and the files had to be split and then the script run on each split file. It is likely that this issue could have been overcome by using the non-generic scatter and gather methods, which allow more customization, but due to time constraints, this idea did not come to fruition.

One challenge that we realized after beginning the analysis phase of the project is that we did not quite handle new line characters "\n" and other whitespace characters well in the cleaning process. We discovered that the body of the posts contain newlines which cause some anomalies in the CSV file. Specifically speaking, some CSV rows which were supposed to form one row were broken into two lines in the cleaned datasets. Thus, we have to handle these abnormal lines in our analysis code. This is a valuable lesson which we will keep in mind in the future.

Exploratory Analysis

Who are the most active users?

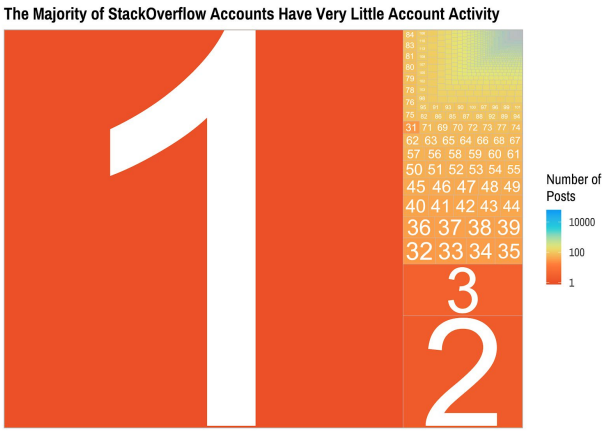


Figure 2: Result of the user activities analysis

The objective of this analysis is to gain more understanding about user activities. To carry out this task, we utilize the Spark framework which we learned in class. Since the Posts.csv file contains data on questions and answers that were posted by each user ID. We were able to do a simple counting of how many times a unique ID appears in the file and extrapolate the user activities from that. We define user activities as the number of questions and answers posted by a user.

Instead of using Spark's resilient distributed dataset (RDD), we opted for a Spark's dataframe to gain more exposure to the data structures provided by Spark. The Spark's dataframe is built on top of RDD, but the advantage of a dataframe is that it allows users to interact with it as a table with rows and columns. This proves to be very intuitive for us since we frequently use Pandas or R dataframe in other classes. The challenge of this task lies not in the computation complexity but in learning the syntax and figure out how to submit Spark jobs on Google dataproc. Learning how to launch a Spark cluster was not as difficult as we thought. In fact, it was fairly easy since we previously learned how to launch and submit a MapReduce/Hadoop job on a dataproc cluster in a lab. This exercise proves to be valuable because we were able to apply skills we learned in class to a slightly different problem.

The runtime of this analysis with 1 master node and 3 workers is approximately 2 minutes 45 seconds which is very fast compared to other MRJob approaches that we did for other tasks. We wrote the code for the same analysis which yields slightly different output (not in terms of discrepancy) using a MapReduce framework. Running the task on a default Google dataproc setting takes about less than an hour. However, we were not able to compare the gain in performance from using Spark to MapReduce/MRJob unless we control for the number of nodes and the location of the server. A more rigorous comparison is definitely needed. Big data frameworks are certainly useful for this analysis because the file is 22.5 GB in size which cannot even be read on to memory or a Pandas dataframe.

What is the most popular question for each year?

The objective of this task is to determined how has the most popular questions asked on Stack Overflow changed over time. This helps us gauge how the developer community has changed over time and what are questions are most likely to be answered.

To accomplish this task, attributes such as post type ID, answer count, question title, and creation date from Posts.csv is used. MapReduce framework in MRJob is used to complete this analysis. In the mapper, only the posts with questions are considered by limiting analysis to posts that have post type ID = 1. For all questions, the creation date, title, and answer count is extracted. The year is extracted from creation date and the mapper yields year as the key and the title and answer count as the value. In the combiner and the reducer, the year, title and answer count of the maximum answer count are yielded. The entire program took about half an hour.

The result of the analysis is shown below. It is evident that with time the type of question has also changed. Earlier, open ended questions with very high number of responses seemed normal and acceptable. However, in the recent years, the questions are specific and seem to target error resolution. It is to be noted, the result is missing value for year 2012. This is due to an anomaly carried over from the data cleaning process.

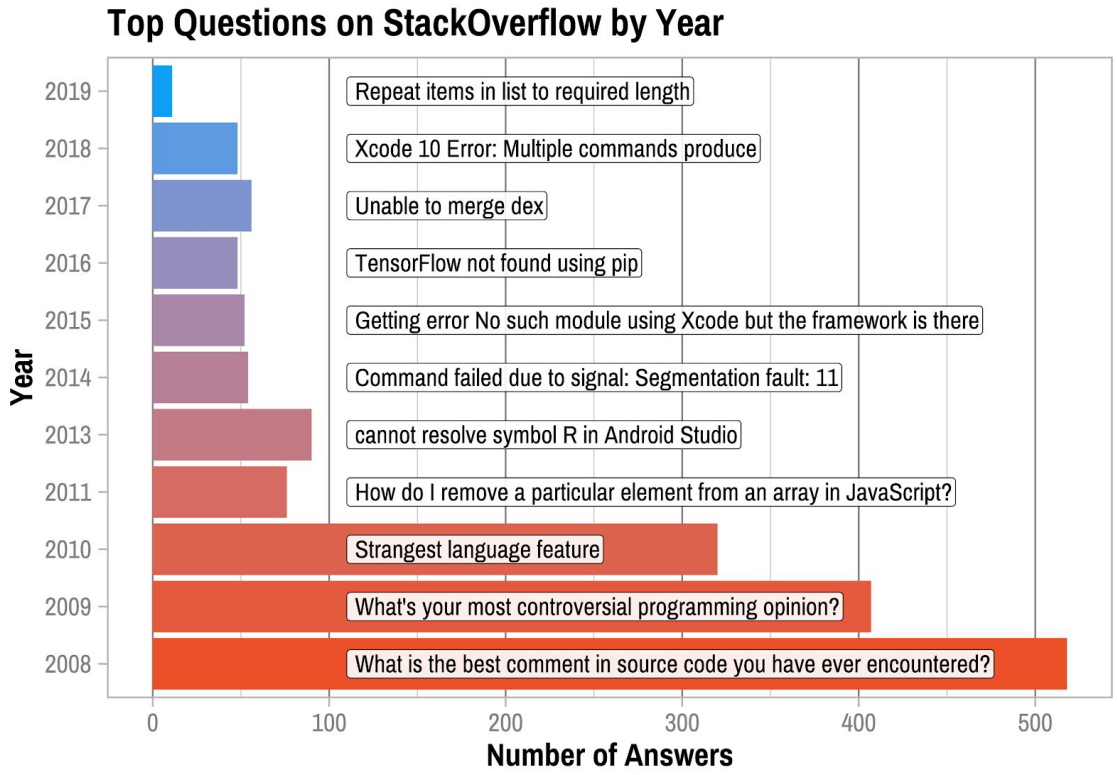


Figure 3: Result of the question analysis

Where do the users that answer the most come from?

In this task, we try to get a sense for the locations of the most active answerers. This information will be helpful to determine if most active answerers are predominantly from English speaking nations or not. Depending upon this, it would strengthen or weaken findings of the main sentiment analysis of responses. If the population of answerers are mostly non-native English speakers, there choice of words and expression may differ considerably.

In order to find an answer to this question, information from two data files, Badges.csv and User.csv, is used. Users file includes information about all users and details about their account. A badge is a commendation given by Stack Overflow to users for achieving various milestones. For most categories, there are three different classes (gold, silver and bronze) and each of them has a unique badge name. Since we are interested in top answerers, we will find the users with the gold badge in answering, named "Illuminator", that is given to users that have edited and answered 500 questions (both actions within 12 hours, answer score > 0).

A multistep process to transform the data and creatively using common keys allowed a relatively simple yet effective MapReduce analysis. In Badges.csv and Users.csv, a file identification, Badges or User is added to the last column of every row using mawk command in linux terminal. Then, these files are concatenated using the cat command and stored in a new file, Badges_Users.csv. This sequence of commands is expedited by encapsulating it in the a shell script.

Having prepped the data, the final results are determined by using a MapReduce framework on the combined file. Based on the file identification, the mapper yields different keys. If the file is identified as "badges", the mapper yields the user_id as key and badge name as value for those users who badge matches the criteria, in this case, illuminator. If the file is identified as "users", the mapper yields the user id as the key and the location as the value. The location is a user entered value and it is not in consistent format. Some users include cities and states while others do not. Sometimes acronyms such as USA is used while other times they are not. To address this variation, all locations are converted to latitude and longitude of country name using geopy package. From that, the geopy reverse is used to convert the latitude and longitude into standardized location and the country name is extracted out. Therefore, the mapper output is a consistent location that can be easily visualized.

The mapper outputs two different key-value pairs, but the reducer takes advantage of the fact that the user id in badges is the id in the users is the same to output the user id key and location and badge as the final output. This process is summarized in the figure below.

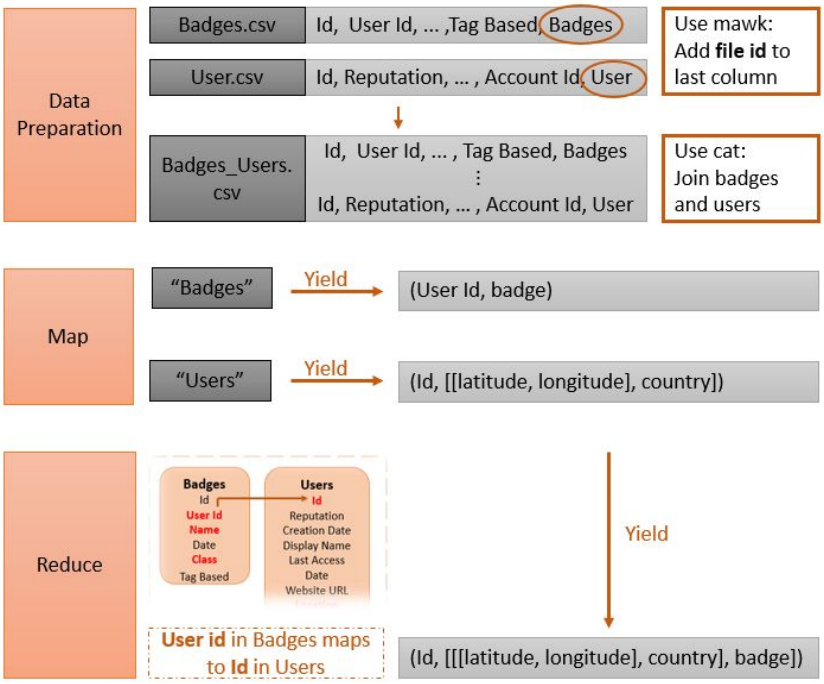


Figure 4: Result of the answer providers analysis

The results of using this process on a smaller subset is visualized in the figure below. From the results of the subset of data, it can be seen that most active answerers come from United States. Due to the low count of users from all other countries represented, it is hardly visible on the map.

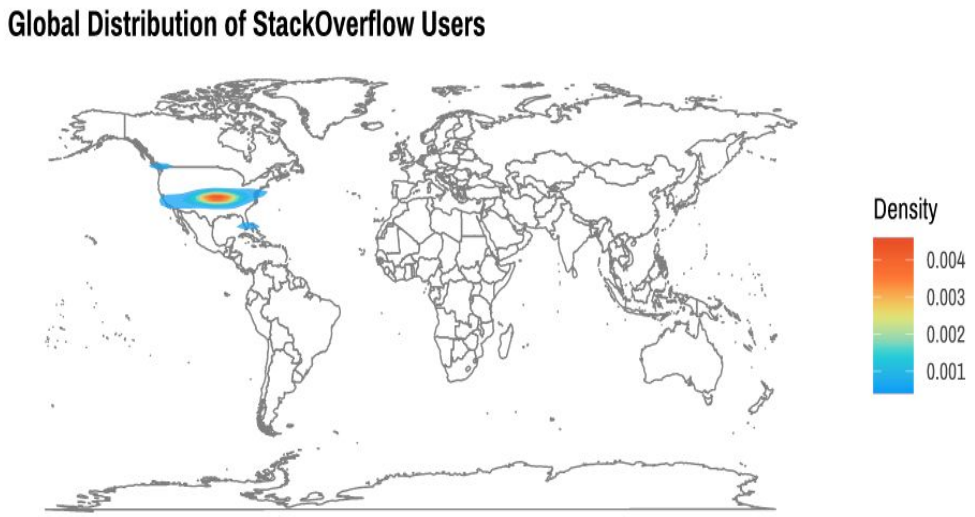


Figure 5: Map Showing Concentration of Most Active Answerers

The major challenge in this task was running the program that uses external python package on the larger dataset using Dataproc clusters. First, a configuration file installing pip and geopy is created and added to the folder. In spite of being able to recognize the external package, the program did not

successfully run for the entire data to output values on the cluster. Additionally, recognizing that runtimes are greatly increased by standardizing the location for every single entry, an updated optimized code is devised that eliminated the location conversion in the mapper and utilizes a multistep MRJob to only convert the locations for the reduced output.

The remaining challenge that we were not able to resolve as of the project submission time is running the multistep optimized code on a dataproc cluster. The code works well on a subset of the data when we ran it locally. In addition, we ran into geocoding request limit which causes our program to stop half way. The latter problem is understandable because we have slightly more than 100 records which require geolocations.

What are the 15 most used tags?

The objective of this task is to sort the tags with the highest count in descending order. This will be helpful in understanding which are the most popular languages tagged on Stack Overflow and some of them will be considered for the main sentiment analysis.

This task is accomplished by using tag name and count attributes from Tags.csv file. The analysis is completed by using MapReduce framework and MRJob package in python. The mapper yields the tag name and count as the key and none as the values. The reducer is initialized with an empty heap. In the reducer, the heap queue is populated up to 15 values with count being the primary key of comparison. After, the top k values, if there is any tag, count pair with a higher count than the first value in the heap, the first value is replaced. Finally, the reducer outputs the count and tag name in descending order.

From the graph below, shows the top 6 languages and we see that javascript is the top tagged language.

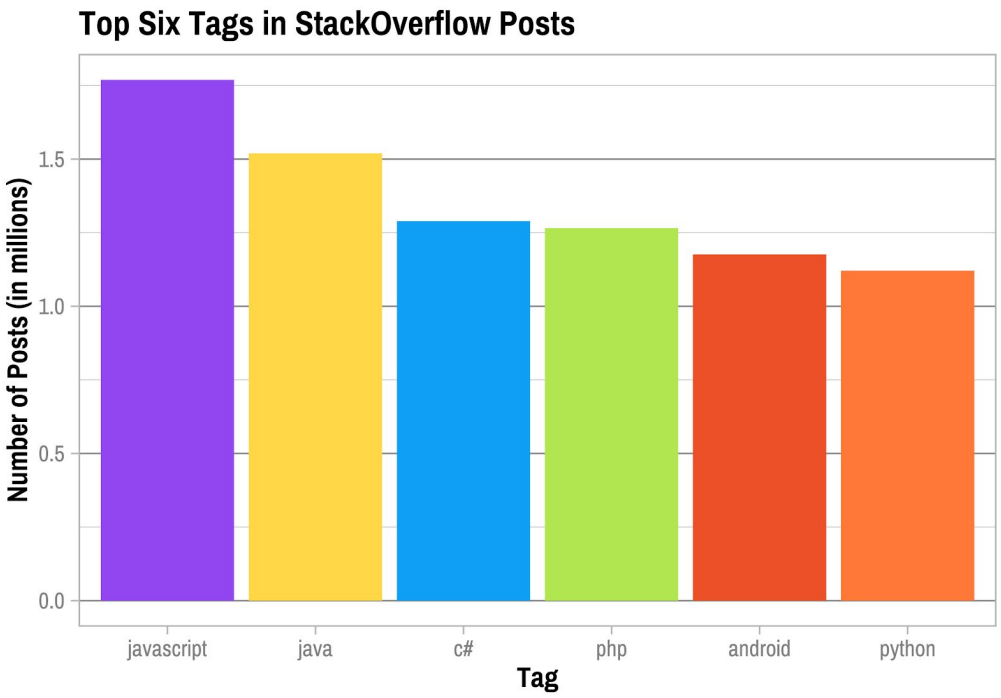


Figure 6: Result of the tag analysis

Which tags are connected?

In addition to finding the most popular tags, we wanted to explore which tags are most often used together and gauge if there is a pattern to the connection.

To accomplish this objective, the MapReduce framework and MRJob in python is used on Posts.csv. From each line in the data, the list of tags is extracted and split to get the adjacent tags. The mapper yields the sorted bigram pair as the key and a value of 1. The combiner and reducer sums the values for a given tuple of tags and yields the pair of tags and the total count. The entire program using the MapReduce framework took about an hour.

The figure below shows the network of tags formed by the output of the program. From this result, it can be noted that the popular language forms the central nodes of the the network.

There is definitely a room for improvement on our implementation of this task. Our method for generating bi-gram is not exhaustive in a sense that it does not generate all possible permutations of tags that are included in a single post. For instance, if the tags are (a, b, c), our approach only finds adjacent tags which would yield merely (a, b) and (b, c). It is noted that (a, c) is not yielded. We could have implemented an exhaustive method using the itertools library, but we decided to limit to adjacent tags because our dataset is large enough that we would still be able to capture the right network structure as shown in the figure below.

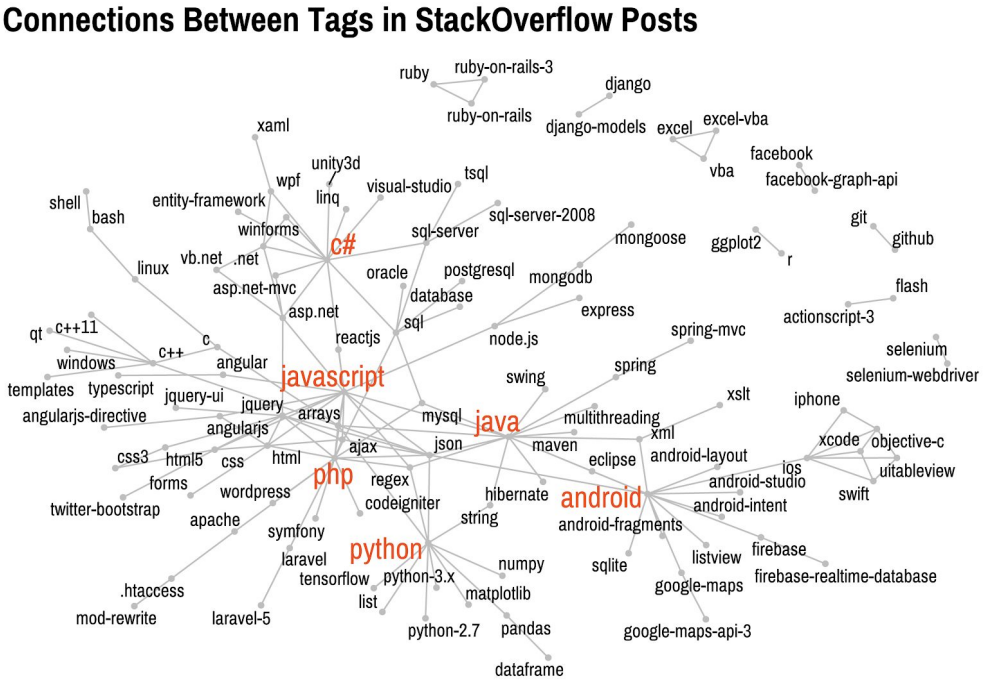


Figure 7: Result of the tag network analysis

Main Analysis

Sentiment & Popularity

In this section, we will answer the question: how do the answerers' sentiment correlate with the popularity of the language? Our hypothesis is that the sentiment would be more impatient and meaner as the language attracts more users. One possible reason is that proficient and professional users do not like to see newcomers of such languages (newbies) post answers that seem trivial to them. We operationalized popularity as two variables in the Posts.csv data: view counts of the questions and the number of questions within the same month (could also be year, quarter, day).

We applied the VADER algorithm (Valence Aware Dictionary and sEntiment Reasoner) to infer the sentiment level in the review, which is a lexicon and rule-based sentiment analysis tool trained with social media data (Hutto & Gilbert, 2014). The final score is a metric for magnitude of the sentiment intensity normalized between -1 and 1. The following table shows how the score corresponds to the sample answers:

Sentence	Sentiment
" This question is stupid..."	-0.57
"You need a closing bracket."	0
"Nice question! You are almost correct."	0.92

Table 2: Sentiments Examples

We designed two steps of MapReduce to extract the popularity and sentiment data for one programming language. Let's take Python for example. In the first MapReduce, the purpose is to get the index of accepted answer ID for the questions with tag "Python". Thus, the key is the ID and value is the view counts of the questions. In the second MapReduce, the key is month and the value is a list of three averaged values: sentiment, view counts, and number of questions. The entire process takes approximately 40 minutes per language. Although we did not attempt to use conventional Python method, it is fair to say that running the program on a single machine without MapReduce framework would have taken much longer.

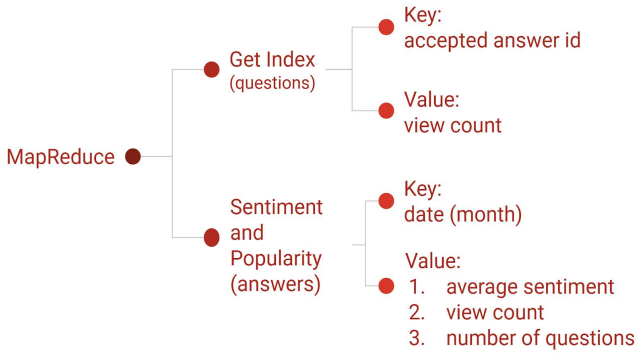


Figure 8: MapReduce Design

We plotted the time-series data from the second MapReduce. For Python, we can see the popularity (view counts and number of questions) has increased steadily in the past 10 years. However, the sentiment fluctuates between months and has a downward trend since 2010.

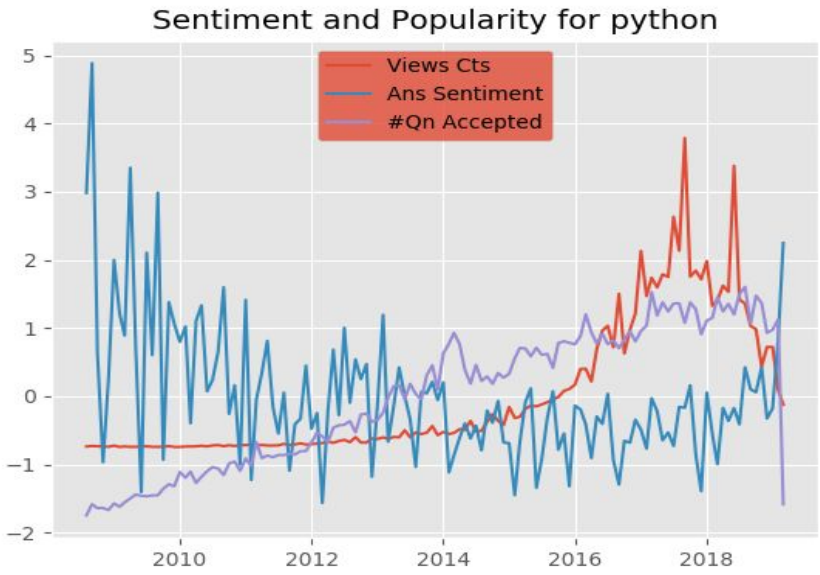


Figure 9: Results of Sentiment Analysis on Python

We also made the time-series plot for other 9 languages that we are interested in:

Row 1 (system tools):	Unix, Dataproc, Git
Row 2 (general-purpose language):	C, Java, Javascript
Row 3 (language with smaller communities):	SQL, R, Rust

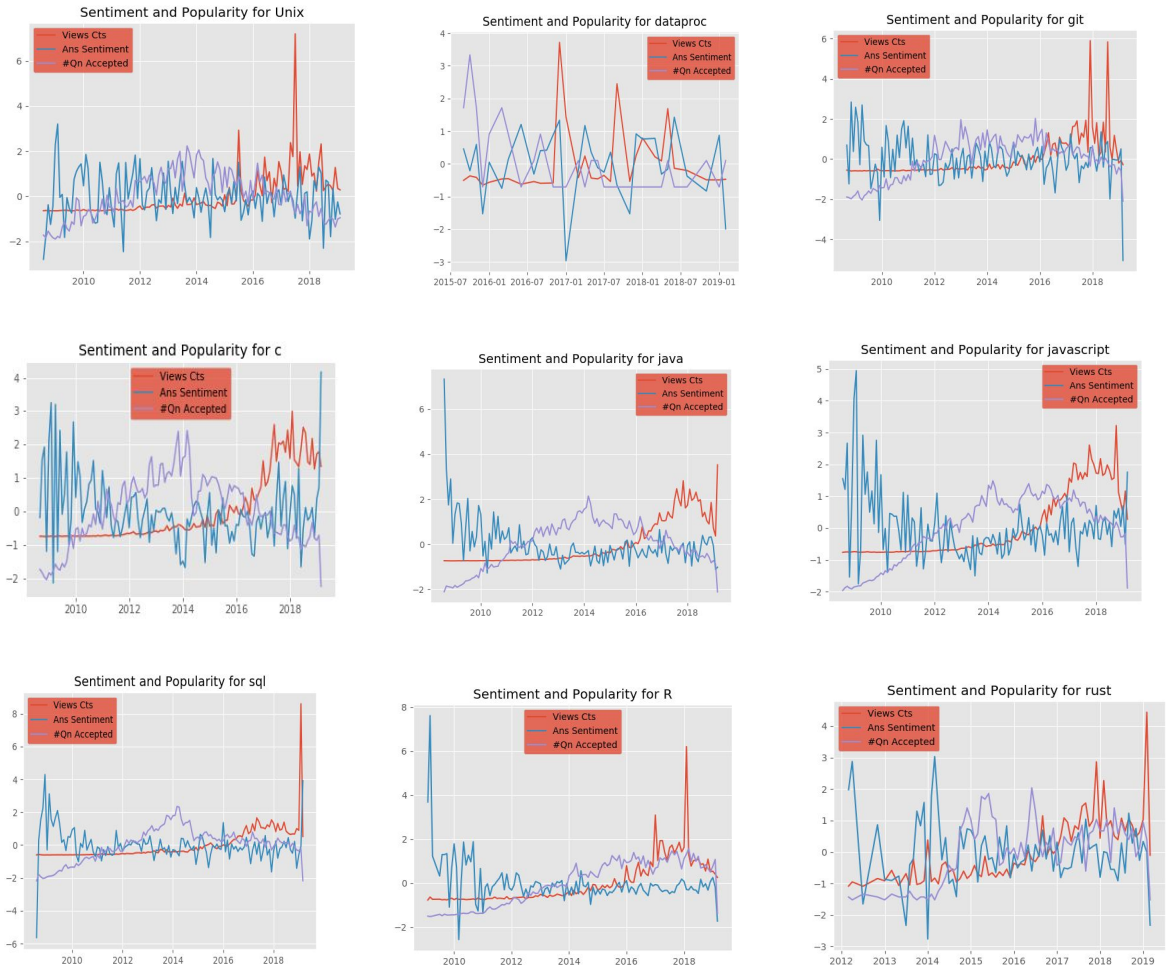


Figure 10: Results of Sentiment Analysis for other 9 languages

To quantify the results, we calculated the correlation between view counts and sentiment as well as number of questions and sentiment. The mean of both correlations are negative, which suggests some evidence for our hypothesis.

Language	View Counts v.s. Sentiment	Questions Counts v.s. Sentiment
C	-0.05	-0.44
Dataproc	0.03	-0.05
Git	0.02	-0.07
Java	-0.2	-0.56
Javascript	-0.04	-0.44
Python	-0.27	-0.5
R	-0.17	-0.32
Rust	-0.06	-0.05
SQL	-0.11	-0.32
Unix	-0.03	0.09
Mean(Std)	-0.088 (0.098)	-0.266 (0.228)

Table 3: Correlations between popularity and sentiment

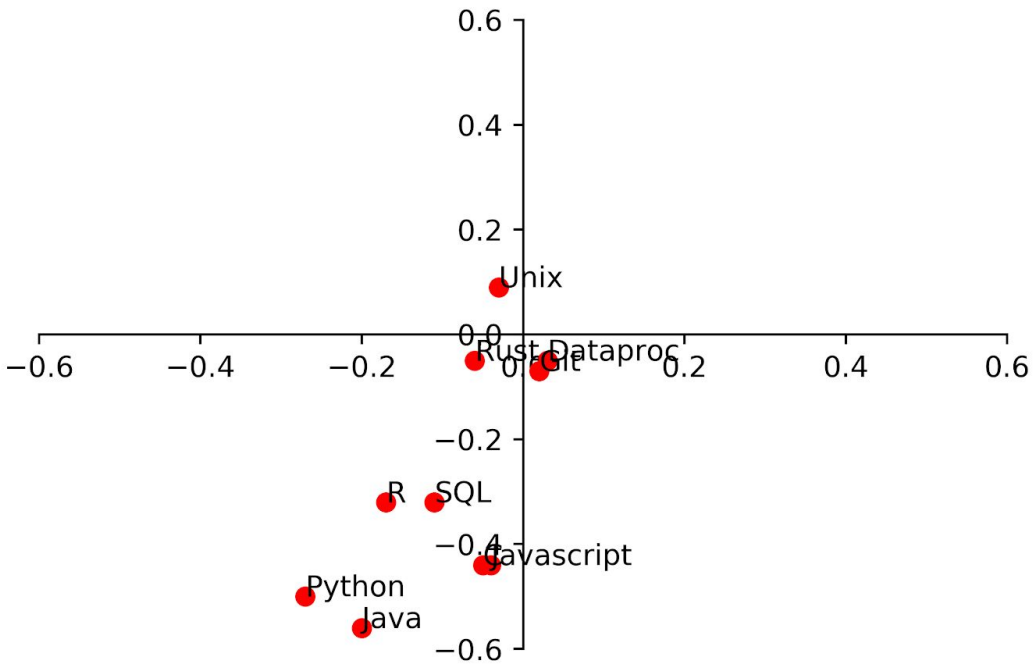


Figure 11: Scatter plot of correlations in the coordinate plane

To visualize the correlation relationships, we made a scatter plot of correlations in the coordinate plane. The plot suggests possible clusters among languages: Python, Java, C, Javascript (general-purpose language) have relatively larger negative correlations and are closer with each other in the third quadrant.

By comparison, Unix, Git, Dataproc (system tools) have relatively insignificant correlation values and locate near the origin.

Analysis Conclusion

One limitation is the reliability of the pre-trained sentiment analysis packages. Although they provide a fast and robust sentiment estimate to the text, language is often ambiguous and they do not perform well in detecting the real semantic meaning behind jokes, sarcasms, metaphors, etc.

Our preliminary results show the correlations between the sentiment and popularity for a programming language are likely to be negatively correlated on Stack Overflow platform. We also find a lot of variations in sentiments and popularity for all the selected 10 languages in the past decade. This suggests that the language user communities are dynamically changing. Among the users for relatively less popular language, they might have more similar background and interests. So, they probably would have more incentive to build a friendly community by answering questions patiently. However, as a language grows mature, experienced users might regard many new questions as naive and simple, since there are plenty of resources elsewhere.

Our project analyzes the developer community of Stack Overflow through the lens of users and languages heterogeneities. Students and young professionals would find the results useful when they decide which programming language to learn and how to get involved in the community wisely. Stack Overflow administors could adopt our algorithms to build the real-time dashboard to track the trends of the languages and provide data-driven insights for the developers.

Project Discussion and Conclusion

In this analysis, our team focused on utilizing all big data tools and frameworks learned in CAPP 30123/ CMSC 12300 to accomplish the task of analyzing Stack Overflow data. This project would not be easily achievable without big data tools. Since the files are large, running the programs on a single machine would have taken excruciatingly long. For the data cleaning process, exploratory analysis and the main analysis, we used MRjob, Hadoop, MPI as well as Spark with the help of Google cloud technologies namely dataproc and compute engines. In addition to the tools learned in class, we spend some time familiarizing with Google storage and the commands to interact with the storage buckets. We also applied DASK (parallel computing package for Python) to estimate the sentiment distribution of all answers and questions in the original data. Finally, to ensure good project management practices such as pipeline of tasks to complete, clarity of task assignment and timely completion of work, we spent some time at the beginning of project to learn and utilize ZenHub, an agile project management tool that works with GitHub.

Reference

Hutto, C. J. & Gilbert, E. E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.