Twitter Analysis - Final Report

Luka Foy

May 2023

1 Abstract / Summary

This project investigated whether a relationship exists between the influence of a Twitter user, and the features of their profile. To perform this analysis, I applied the PageRank and cosine similarity algorithms to the Stanford Twitter SNAP Dataset [1], which contains data on over 80000 Twitter users. Instead of links between webpages, PageRank will use follows/following between users, which in our dataset, generates a graph with over 2000000 edges. From here, the features of the most influential users' profiles will be examined with cosine similarity. Hopefully, we will see a significant portion of these influential users having high cosine similarity values - as this would indicate there may be a relationship between influence and profile features. A result of this nature would be significant to businesses/individuals who are seeking to develop influence on Twitter, as it would help to inform them on the nature of content associated with influence.

2 Introduction

2.1 Background

My chosen dataset is the Stanford Twitter SNAP Dataset[1], which contains data of 81306 unique twitter users, contained in a .gz file. It includes 5 record files for each user (the user whom the record is centered on is called the "ego user" in the context of the dataset). The 5 records are as follows:

1. The circles file contains clusters of users connected with the ego user. I will ignore this for the purpose of this project, as the data it contains is not relevant to our analysis.
2. The edges file contains a set of graph edges (stored like tuples) for people who the ego user follows on twitter. These edges are directed, (a follows b), and we assume the ego user follows all users contained within this file.
3. The featnames file gives all the names and types of features which appear in the profiles of ALL users listed in the edges file. Features, in the context of Twitter, are hashtags and @ mentions of other users which are found on a profile.
4. The egofeat file is a vector of 0s and 1s which correspond to the ego user's profile features. 1 meaning that the user has this feature in their profile, and 0 meaning they don't.
5. The feat file is a set of vectors for each of the users which appear in the edges file. This data is the same as that in the egofeat file, in that 1s mean the user has the feature in their profile, and 0s mean they don't.

I am using both the PageRank and cosine similarity algorithms. PageRank determines the relative "importance" of an item within a network of other items. To do this, we take a graph as an input, and initialise all the nodes in the graph to a PageRank score of 1. Then, we create a teleportation matrix, which maps the probability of a person surfing the web clicking on a link (this probability is (1 - damping factor), which we will set to 0.15). This prevents both the spider trap and dead end problems, which will be explained in more detail in the Design & Methods section. Next, we divide the PageRank of a user by the number of external links it has (in this case, the number of other users the user follows), and these fractional values are distributed to those users who are followed by the original user. We then sum the matrix values and the contributions from other nodes to generate the user's PageRank. This process is repeated until the scores converge, at which point our PageRank is complete, and the users with the highest scores represent the most influential users in the network. Cosine similarity measures the likeness of two non-zero vectors. It's computed by taking the dot product of two vectors, then dividing by the product of their lengths. The result yields a value between -1 and +1, +1 meaning that the vectors are identical, 0 meaning that they are orthogonal, and -1 meaning they are the complete opposite.

2.2 Motivation

This research question is relevant to me as I have always been fascinated by the metrics which drive user engagement on social networks. To be able to work with real Twitter data and potentially draw meaningful conclusions is very motivating. Out of the algorithms we have covered, PageRank seemed like the biggest challenge for me personally to implement, hence why I have chosen it as my main algorithm, so I can gain a deeper understanding of parallel programming. The application of PageRank to a social media dataset seemed like a logical fusion of ideas, and was inspired partially by this Medium article. However, I didn't just want to apply PageRank, I was looking to extend my analysis of the dataset to draw a more meaningful conclusion than simply ranking Twitter accounts based on influence. Therefore, I am applying cosine similarity as well, to attempt to identify a relationship between the most influential Twitter accounts. As stated in my Abstract, the outcome of this project may be useful for both people driven by curiosity, or individuals / businesses who are looking to develop an influential Twitter account. If there is a relationship between page features and influence, this could guide the content they choose to post.

2.3 Research Question

My chosen research question is "Is there a relationship between the profile features of a Twitter user, and their level of influence?". More specifically, we will investigate the relationship between profile features of the top 10 highest scoring ego users from PageRank.

The dataset is comprised of various data Twitter users' profile features and connections to each other, hence the question's relevance. The PageRank algorithm will use the connections between user profiles (through "follows") to help us to determine which users are highly influential. Then, cosine similarity will help us to determine how similar pairs of profiles are with each other, based on their features. Thus, it will help us to determine whether there may be significant relationships present between the influential Twitter users.

3 Experimental Design and Methods

3.1 Preprocessing - PageRank

As importing and transforming raw data into a format that is usable for algorithmic analysis can be very time intensive, I decided to do the majority of my preprocessing beforehand. My raw data file from the Stanford website was in the .gz, or GnuZip, format, and contained five different text-like files for each ego user in the network. Within the processing document, I used !wget to download the file directly from the Stanford website and into a directory in my Google Drive. The relevant graph files for PageRank had the ".edges" extension, thus I used a for loop to go through all the files in the directory and get the data paths of files with ".edges" as an extension, and add them to a list of other file paths with the same extension [2]. Then, I used a list comprehension to read in each ".edges" file into a Spark dataframe, and stored the file path as a column within the dataframe [3]. Finally, I concatenated all the dataframes together using the ".union()" method, and converted it into a RDD - which I stored as a text file in my mounted Google Drive. Using this implementation, I can import the text file using a call to the ".textFile()" method of a SparkContext in my main PageRank notebook.

3.2 PageRank Implementation

After pre-processing my data, the import was very simple - and my code from lab 4 only had to be modified slightly to read in the graph data into an RDD of (source, [destinations]). From there, I used my full lab 4 implementation as a starting point, with the aim of increasing algorithm efficiency and accounting for dead ends and spider traps in the graph. To help increase efficiency, my code caches the input file to the main function, as we are referencing it in up to 100 iterations throughout the call to page_rank_main(). Also, I added checks for convergence - in addition to the static number of iterations, with the hope of potentially reducing redundant iterations if the PageRank scores were converging earlier than expected. In order to account for the un-connectedness of the graph, I used a damping factor of 0.85 - which models the probability that a user will continue browsing through links (or through a user's "Following", in this case) as opposed to randomly visiting another user's profile [4]. This is applied to the PageRank scores in our loop, by using a call to reduceByKey() (after obtaining the R vector of scores) to sum the contributions from incoming user follows, and incorporate the damping factor. Convergence checks were done by comparing the current and previous version of R, and simply checking the sum of the difference between PageRank scores for each node.

Other than the above, my implementation is the same as the one I developed in lab 4, see the attached screenshots for the full implementation, including helper functions. Please note, the output was filtered to only include the top 10 most influential ego users, as the feature data was tailored specifically to these users as opposed to other - non ego users. However, these other users were included for the PageRank analysis.

3.3 Preprocessing - Cosine Similarity

Starting similarly to PageRank, using !wget to download files to store files fron the Stanford website directly into my Google Drive. For the purpose of my cosine similarity implementation, we need all the featnames files and all the egofeat files. I started by defining a function to read all the files of the specified type into dataframes, which were then concatenated together (using the Spark dataframe .union method) and converted into an RDD (using the .rdd method). This gives us two RDDs, one containing the names of features of profiles followed by the ego user, and the other containing bit vectors which display whether or not that feature is in the ego user's profile (0 if not, 1 if yes). From here, I transformed the featnames RDD into a usable format by performing a series of splits, removing problem characters from the feature strings, filtering the users (to only include the top 10 users from the PageRank analysis) and grouping feature names by user (.groupByKey() method). At this point, the RDD contains a list of each feature within a user's profile. Then, the file is saved into my drive, ready to be imported by my cosine similarity notebook.

Our other desired input for cosine similarity is a sorted dictionary containing every influential user's features. So, to do this, I again performed a series of splits and filtered the non-influential users out. Next, I joined the two RDDs, and created a dictionary for each user, with feature names as keys and bits as values (representing whether or not they were present in that user's profile). From here, I filtered all features that were not present (based on whether their bit was a 1), and used flatMap(), distinct(), zipWithIndex() and collectAsMap() to return a dictionary of all distinct features that are present in any of the profiles of the most influential users. I then converted this dictionary into a dataframe with one concatenated key:value column (using PySpark's concat_ws method), to be stored into my drive.

All the RDD transformations were quite time-consuming, hence why I decided to include them in my pre-processing step instead of including them in my cosine similarity notebook. Now, we can use a combination of the names of features in a user's profile - along with the dictionary of all distinct features to produce SparseVectors for each user (representing the same set of vectors) that can be compared between users with cosine similarity.

3.4 Cosine Similarity and Visualisation

After preprocessing and storing my dictionary dataframe and RDD of features and their assoicated users, I read them into the main computation file for my cosine similarity implementation. I then converted the dictionary dataframe back into a dictionary by using the selectExpr() dataframe method to put the keys and values back into their correct columns, row by row. I formatted the featnames RDD using some .split()s and .strip()s to remove unwanted whitespace and other characters.

For the cosine implementation, I followed James' recommendation of using SparseVectors to reduce the amount of computation required, and cached my RDD containing the sparse vectors for each user. These represent whether or not the feature (the ones contained in any influential user's profile) is present in that specific user's profile. To perform all the comparisons, I looped through every influential user, using one as the baseline every time to perform comparisons against the others using .map()s. The cosine similarity values were outputted into a list of tuples containing the two users who were compared, and their similarity score.

From here, I simply generated the proportions of comparisons that were above and below the threshold, and used matplotlib's plt.bar() function to visualise those proportions.

3.5 Libraries and Modules

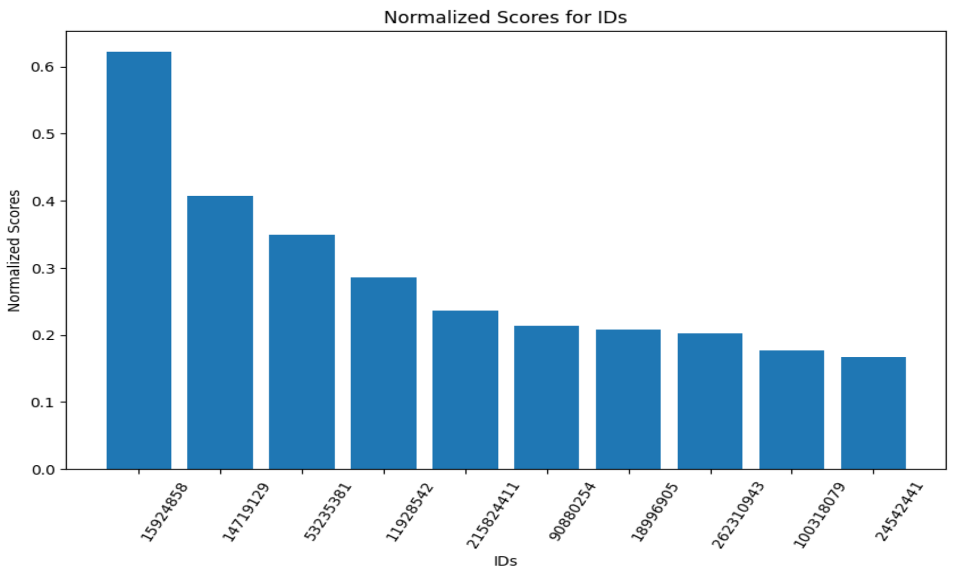
- Vectors - module containing SparseVectors, used for cosine similarity.
- NumPy - used for cosine similarity calculations and visualisations.
- matplotlib - visualising results.
- sc.textFile - Reads .txt file into a PySpark RDD.
- spark.read.text - Reads .txt file into a PySpark dataframe.
- math.dot(), math.numNonZeros - for cosine similarity calculations.
- google.colab.drive - allowed me to access datafiles in my Google drive for imports.
- .persist(), .broadcast() and .unpersist() to save time communicating RDDs between processors.
- os.listdir() to get files from a directory.
- If I included a list of all functions I made, this would be very long. Thus I've just specified in detail the implementation above, I trust this will be enough to understand my methods. All code in notebooks has been fully commented.

4 Results

4.1 Relevant Figures

4.1.1 PageRank

The graph below represents the user IDs and normalised PageRank scores of our 10 most influential ego users. We can see the normalised score of the highest user was almost double that of the next highest, however they smoothed off at the end. Given the number of edges in our dataset, we would expect minimal differences between the lower-scoring users, as there were very few high-value scores to be gained (most users in the dataset followed a large number of other users, thus making their outgoing scores less significant).

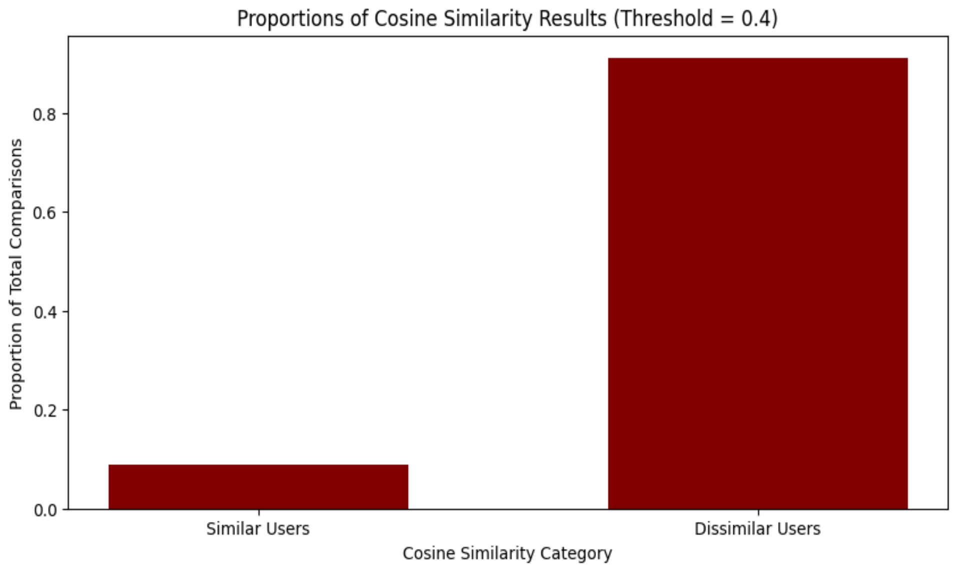


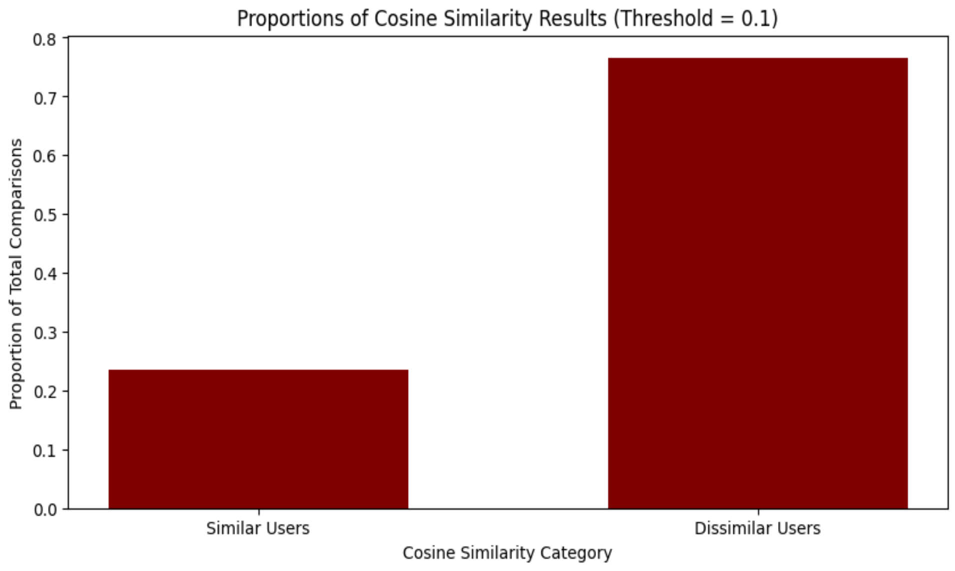
As explained in our method section, the features of these 10 most influential users were then used to calculate the cosine similarity scores below.

4.1.2 Cosine Similarity

After running my implementation of cosine similarity on the 10 most influential ego users in the dataset, I produced the following graphs to represent the proportion of comparisons which yielded similar results. I am using a significance threshold of 0.4, as in our context we are are not overly concerned with finding very similar users, but merely trying to establish if there may be a relationship in profile content present between influential users.

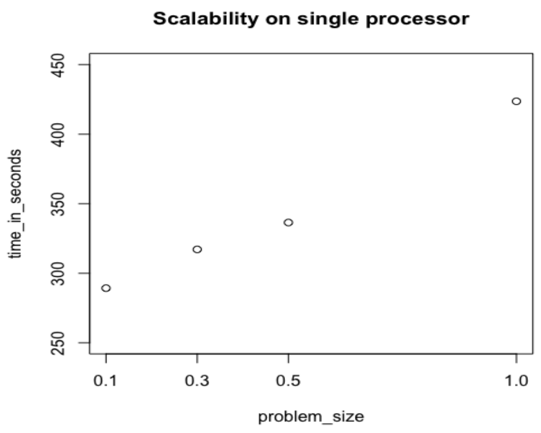
The graph with a threshold of 0.1 has just been included for reference.

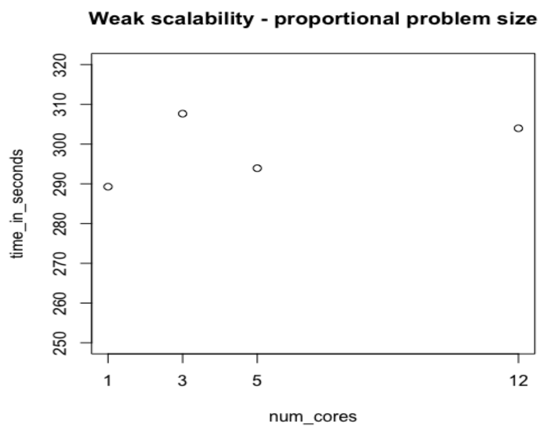




4.1.3 PageRank Scalability

I ran the two different sets of testing on my PageRank implementation. In order to vary the problem size, after preprocessing my RDD, I took different sized samples of it (using RDD.sample()) in order to simulate a change in problem size. The following graphs show what I observed (note problem_size for the single processor scalability is measured as a proportion of the size of my main dataset, bsaed on how big the samples were):





4.2 Discussion

Our visualisations above indicate a proportion of roughly 10% of influential user pairs with cosine similarity greater than 0.4, and roughly 90% less than 0.4. This suggests that there is limited similarity between the profile features of the top 10 most influential users in our dataset.

Since the cosine similarity values are generally low, it implies that the level of influence of a Twitter user may not be strongly correlated with their profile features alone. Other factors, their engagement with other users, and influence on other platforms, could potentially play a more significant role in determining their influence.

To conclude, based on the provided cosine similarity output, there seems to be limited similarity between the profile features of the most influential users in our dataset. This suggests that there may not be a significant relationship between profile features and user influence on Twitter, indicating that other factors likely contribute to a user's level of influence.

5 Conclusion

To conlcude, we were able to answer the research question. Our analysis found only 9% of comparisons between the most influential profiles yielded a cosine similarity greater than our threshold of 0.4. This suggests the relationship between the features of a user's profile and that user's influence is relatively minimal. If a relationship were to exist, we would expect a far larger proportion of our comparisons to yield a significant cosine similarity.

However, the implications of this result for businesses / individuals who are interested in what drives the influence of a social media account may still be useful. This demonstrates that the nature of the a user's profile features may not be correlated with how much influence that user has. Thus, there are likely numerous other external factors which contribute to this influence. I would argue this is still a positive implication, as it provides businesses / individuals with context around influence and suggests that profile features are not directly responsible for it. This could spark their curiosity and lead them to investiage the topic further.

If I were to take this project more in depth, I would love to expand the size of my cosine similarity dataset - and potentially include a term frequency algorithm like TF-IDF to determine which features in particular were most associated with the most influential profiles. Furthermore - although outside the scope of this curse - I'd also consider taking it in a more statistical direction with regards to analysis - and potentially develop some models which predict influence based on somewhat similar parameters to this dataset.

6 Project Critique

Choose one part of your design or methods (section 4 of your proposal, the algorithm, data flow, and software) that you feel could have worked better with a different approach. Describe what did not work with your design or method. Why did it not work? What changes could you have made that might have been better? Suggested length: two paragraphs. If anything about the design of this project were to have worked better, I would (as discussed in the conclusion above), would have liked to perform cosine similarity on a larger dataset of users. However, the nature of my implementation (a large number of pair comparisons), meant that this operations was very slow. Potentially, the analysis could be improved a more efficient algorithm to calculate cosine similarity - which allows us to do a large number of comparisons. However, I am still happy with the outcome of the analysis.

Other than this, my project ran relatively smoothly - and I accomplished everything I set out to do. Thus, this section will be more of a discussion about what factors allowed this project to run smoothly, with an emphasis on what it has taught me. Firstly, the main reason this project ran smoothly was early planning and engagement with my topic. As outlined in my proposal and progress report, I developed a week-by-week plan right at the start of the term, and stuck to it. I tried to overestimate how long implementations would take, so it would give me more time to debug if needed. This plan allowed me to stick to the exact same algorithms, dataset and research question throughout the whole project. Secondly, prioritising this project over other university work allowed me to dedicate more time to testing, debugging and developing solutions. This, again, meant all software was more or less implemented as planned.

7 Reflection

Throughout the project, I mainly found the course concepts of map, filter, and groupByKey() functions to be extremely useful. These concepts were covered extensively in our course labs, allowing me to gain hands-on practice and solidify my understanding. In particular, Lab 4 was especially relevant to my project as it involved James' implementation of cosine similarity and the usage of sparse vectors. Using ChatGPT - just for concepts, code was adapted from Stack Overflow and various articles as shown in references. This was useful when Jamie and James were not available to explain concepts, and it helped me gain a better grasp on the algorithms I was using, making it easier to step through them sequentially to generate my code. Moreover, my project emphasized to me the importance of code efficiency and pre-processing techniques to reduce computation time. Also, I feel as though the format of the project assessment - specifically timing and content required for the progress report and proposal - helped guide me to planning a manageable workload. Thus, it taught me the importance of planning, which ultimately contributed to the successful completion of the project.

8 References

I haven't directly worked with any other students on this project. As stated in my code comments, I used James' cosine similarity implementation which he emailed around (using sparse vectors and NumPy calculations) - other than this, all my implementations were adapted from ones I had developed in the labs.

Here are resources that I used to gain inspration for my project, help me to understand algorithmic concepts in more detail, and explain useful PySpark functions:

- [1] Stanford Twitter SNAP Dataset
- [2] os.listdir() Explained
- [3] Stack Overflow files into dataframe
- [4] Explanation of damping factor Medium article - project inspiration
- Example PageRank implementation
- Similar project concept, helpful for gaining context
- Explanation of PageRank
- Inspiration for applying cosine similarity to social media data
- Example cosine similarity implementation
- Help for structuring research question