Analyzing Twitter Data using GraphFrames and PySpark

Anar Kuatzhan
University of Waterloo
akuatzha@uwaterloo.ca

Shriya Vaagdevi Chikati
University of Waterloo
svchikat@uwaterloo.ca

ABSTRACT

This project aims to delve into the "Social circles: Twitter" dataset using Spark and the GraphFrames library to construct and analyze the social network's graph representation. Our goal is to extract valuable insights from the network's structure by identifying key metrics and patterns. Leveraging GraphFrames functionalities and coursework knowledge, we seek to explore various aspects of the network.

Our analysis encompasses several crucial areas, including identifying influential users, uncovering connected components, and detecting communities within the network. Additionally, we visualize a portion of the network and employ machine learning algorithms to gain deeper insights into its behavior. Through this project, we showcase the GraphFrames library's prowess in dissecting intricate social networks.

Keywords: *GraphFrame, Twitter Networks, PySpark, Distributed Computing, Machine Learning*

1 INTRODUCTION

Social media platforms have changed the way we connect with others, interact with the world, share thoughts, and build communities. Twitter, being one of the most influential and diverse social networking platforms, makes it easy for users to interact and connect with each other and gives a better insight into the behaviors of communities. Analyzing social media data is essential for understanding trends, identifying influencers, and detecting patterns that shape online communities.

This project focuses on leveraging advanced techniques in data analytics to explore the 'Social circles: Twitter' dataset. By utilizing Spark and the GraphFrames library, we aim to construct a graph representation of the Twitter social network and extract valuable insights from its structure. Additionally, we will explore machine learning algorithms to gain deeper insights into network behaviors and dynamics. Overall, this project aims to demonstrate the power of data analytics in uncovering meaningful patterns and insights from social media data.

The paper is structured as follows: We begin by introducing the dataset, followed by a discussion of our pre-processing steps. Next, we outline our chosen methodology for conducting the analysis. The Analysis section is divided into two subsections: General Analysis, covering basic operations of GraphFrames, and Enhanced Analysis, focusing on advanced algorithms of GraphFrames and supervised Machine Learning. Following this, the Evaluation section assesses the performance of distributed implementations against non-distributed ones, as well as the machine learning task. Finally, we conclude the paper with a conclusion of our findings.

2 DATASET

We obtained the dataset, which is called 'Social circles: Twitter'[2], from the Stanford Network Analysis Project (SNAP) by Julian M. and Jure L., which was obtained from public sources and the link to the same can be found here: https://snap.stanford.edu/data/ego-Twitter.html. The size of the 'Twitter' folder that we obtained is 277,960,794 bytes or about 265 MB. This folder contains five different types of files for each user (with a unique user ID) with the extensions: '.circles', '.edges', '.egofeat', '.feat', and '.featnames'. Figure 1 shows the structure of the files in the dataset.

The information contained in each of the types of files, obtained from the readme file in SNAP Twitter Circles dataset and the link to it: https://snap.stanford.edu/data/readme-Ego.txt, is as follows:

- nodeId.circles: The set of circles for the ego node. Each line contains one circle, consisting of a series of node ids. The first entry in each line is the name of the circle.
- nodeId.edges: The edges in the ego network for the node 'nodeId'. Edges are directed (a follows b) for Twitter. Each line consists of two node ids. The 'ego' node does not appear, but it is assumed that they follow every node id that appears in this file.
- nodeId.egofeat: The features for the ego user. Features are '1' if the ego node has this property in their profile, and '0' otherwise.
- nodeId.feat: The features for each of the nodes that appears in the edge file. Each line starts with a node Id followed by 0s and 1s.Features are '1' if the node has this property in their profile, and '0' otherwise.
- nodeId.featnames: The feature names for each of the ego users. The number of feature names is the same as the number of ego features and the number of features for different nodes corresponding to each ego user.

Figure 1: The structure of the files in the SNAP Twitter Circles dataset for the ego users with IDs 12831 and 78813.

3 DATA PRE-PROCESSING

The dataset is composed of multiple files, with five distinct files associated with each ego node, as outlined in the previous section. So, all the files cannot be read the same way. To tackle this, we have defined five different functions using Python and Spark to read each of the different files. In order to read each of the files, the functions we defined in Spark performed relatively slow compared to those written in Python. This is due to the overhead of initializing the Spark context and distributing the tasks across the cluster which makes the overhead more noticeable when dealing with a large number of small files. In contrast, using Python code could help avoid overhead associated with distributed computing frameworks like Spark.

The number of files, in this dataset, is large while the contents in each file is relatively small. Even though, the Python implementations were faster for our task, we used Spark because it can potentially handle much bigger files and scales well because it runs in parallel. The files were of the form egonodeId followed by an extension. For each of the files, we extracted the ID of the ego node and then, processed each of the types of files as follows:

- nodeId.featnames: The feature names are processed in such a way that they only contain alphanumeric characters, in lower case, and underscores. A dictionary, with the indices of the feature names as keys and the feature names themselves as values, is created. Since each ego user has different list of features, we created a dictionary with the ego user as the key and the feature names dictionary as the value.
- nodeId.circles: The function returns a list of tuples containing the ID of the ego node, the circle number, and the node ID in the corresponding circle.
- nodeId.edges: The function returns a list containing tuples of the form (node1, node2) and it is assumed that node1 follows node2. Each line returns three tuples, (egonodeId, node1), (egonodeId, node2), and (node1, node2), since it is assumed that the ego user follows all the nodes in the file.
- nodeId.egofeat: The function replaces all the 1's with the feature names in that index, using the dictionary corresponding to the ego user. The duplicate feature names are then removed and tuple with the ego user ID and list of features is returned.
- nodeId.feat: The function behaves similar to the function processing files with '.egofeat' extension but does so for all the nodes in this file.

After defining the necessary functions to read and process the files, we processed the files with specific extensions in parallel. This is crucial because the number of files in this dataset is large and processing them sequentially would be quite inefficient. And so, this method processed the files concurrently across the Spark cluster to manage data processing effectively. We then union the egofeat RDD and the feat RDD to get the node IDs and their corresponding features in a single RDD.

Then, we created dataframes using each of the RDDs to make it suitable to conduct further analysis of the data, and then dropped the rows corresponding to users having no feature names, followed by merging duplicate nodes with different lists of feature names. We then, proceeded to analyze the processed dataset.

4 METHODOLOGY

In this project, we use GraphFrames, an Apache Spark-based distributed graph processing framework, to delve into the complex structure of social circles in our dataset. The efficiency of GraphFrames' built-in functions varies depending on their complexity: simpler operations are typically quick, while more complex ones can take considerably longer. However, it's worth noting that despite this variability, all built-in functions of GraphFrames operate in parallel, leveraging the distributed computing power of Spark. To handle these performance differences and adapt to the constraints of our Google Colab environment, we've adopted a dual approach methodology for our analysis.

General Analysis with Basic Operations of GraphFrames

We take advantage of GraphFrames' efficient handling of basic graph operations like vertex and edge manipulations, which are notably quick even with large datasets. These operations form the foundation of our basic graph exploration tasks, such as analyzing node degrees.

Enhanced Analysis with Advanced Algorithms of GraphFrames

For more complex analyses that require advanced graph algorithms and heavy computational efforts, we strategically optimize our resource use. This approach enables us to conduct detailed analyses, including community detection, PageRank, triangle counting, and link prediction, while effectively managing computational load.

By balancing simple and complex operations, we strive to achieve a balanced mix of efficiency and thorough analysis in our exploration of Twitter social circles.

5 ANALYSIS OF THE DATASET

We conduct two types of analyses: a general analysis spanning the entire graph, and a more computationally intensive enhanced analysis concentrated on a sub-graph, leveraging the built-in functions of GraphFrames [4][5]. Figure 2 shows the sample vertices and edges of the graph.

```
+---+--------------------+   +--------+--------+
| id|            Features|   |     src|     dst|
+---+--------------------+   +--------+--------+
| 17|[obviouscorp, sti...|   |17137927|56845340|
|107|      [jason, ilona]|   |17137927|14116024|
|190|[sfgiants, edcase...|   |56845340|14116024|
|224|[joshuatopolsky, ...|   |17137927|62646404|
|246|[maxfenton, ftrai...|   |17137927|19126166|
|257|[portland, bigsco...|   |62646404|19126166|
|414|[, davepell, rcar...|   |17137927|12661142|
|418|[dens, smithsonia...|   |17137927|  972651|
|528|[irondavy, bradel...|   |12661142|  972651|
|586|[jess, anildash, ...|   |17137927|15859039|
+---+--------------------+   +--------+--------+
```

Figure 2: Sample vertices and edges

5.1 General Analysis: Analyzing the Entire Graph with Basic GraphFrames Operations

5.1.1 Connected-Components. By utilizing the 'connectedComponents' built-in function of GraphFrames, we can identify distinct components within the graph. Grouping the obtained components by size reveals a significant component encompassing 68,262 nodes

out of 70,923. Conversely, the remaining components are single-node entities. To refine the graph, we use the 'dropIsolatedVertices' functionality of GraphFrames to eliminate these isolated vertices. Consequently, we are left with a single large component containing 68,262 nodes interconnected with 6,860,727 edges. Figure 3 provides a glimpse of the large component we've been analyzing. Note that some nodes appear isolated in the visualization. This occurs because we've sampled 1,000,000 nodes from the entire graph as visualizing the entire graph isn't feasible due to our limited resources on Google Colab.

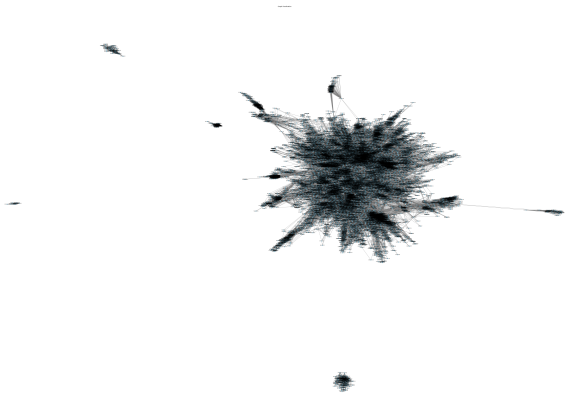

Figure 3: Sample graph visualization

5.1.2 Node Degree Distributions. We examine the distribution of node degrees in the graph by utilizing the functionalities provided by GraphFrames: 'degrees', 'inDegrees', and 'outDegrees'. The resulting tables, displayed in Figure 4, present the top 10 rows of the complete tables.

degree	count
2	3477
4	3695
6	2897
8	2868
10	2487
12	2405
14	2081
16	2047
18	1821
20	1717

id	inDegree
40981798	18695
43003845	16805
22462180	15793
34428380	15655
3359851	11183
115485051	9383
11348282	8861
7860742	8734
15913	8593
7861312	7917

id	outDegree
256497288	35869
314316607	33206
16987303	27100
217796457	25109
307458983	21018
24117694	19657
200214366	19481
89826562	19438
187744415	19278
2097571	18851

Figure 4: Node degree distributions

Degree Distribution. The leftmost table in Figure 4 displays the frequency of nodes with different degrees.

The distribution appears skewed, indicating a larger number of nodes with lower degrees compared to higher degrees. This suggests the presence of a core group of highly connected nodes surrounded by a greater number of less connected nodes, which aligns with the analysis of connected components. Nodes having higher degrees potentially serve as bridges connecting various parts of the network.

In-degree Distribution. The middle table in Figure 4 showcases the top 10 nodes with the highest in-degree.

Node 40981798 boasts the highest in-degree of 18695, followed by node 43003845 with an in-degree of 16805, and so on. These nodes likely play significant roles in receiving connections from other nodes in the graph, and are presumed to be associated with the most influential users within the entire network.

Out-degree Distribution. The rightmost table in Figure 4 exhibits the top 10 nodes with the highest out-degree.

Node 256497288 leads with the highest out-degree of 35869, followed by node 314316607 with an out-degree of 33206, and so forth. Nodes with high out-degrees are likely to be the most active users, particularly in terms of following others.

5.2 Enhanced Analysis: Leveraging Advanced Algorithms of GraphFrames on a Sub-graph

To explore the full potential of GraphFrames, we'll focus on a sub-graph formed by randomly selecting edges and their associated vertices. This approach is essential given the vast size of the original graph, which contains 68,262 interconnected nodes and 6,860,727 edges. While GraphFrames' parallel processing capabilities enable efficient operations across the entire graph, directly analyzing such a large graph would require substantial memory and processing time, especially within the constraints of platforms like Google Colab. By using a sub-graph, we can effectively demonstrate the diverse capabilities of the GraphFrames framework while staying within our computational limits.

5.2.1 Connected-Components. This sub-graph, created by randomly selecting edges, displays a higher number of connected components compared to the original graph, which is dominated by a single large component. Among these, two significant connected components contain 385 and 309 nodes, respectively. Additionally, there are two other components, each with 47 nodes. The rest of the connected components are relatively small.

5.2.2 Triangle Counting. The triangle counting analysis on the sub-graph reveals the presence of triangles, which are closed loops connecting three nodes.

Each count in the table in Figure 5 signifies the number of such closed loops associated with the respective node. Higher triangle counts suggest a more complex network of interconnected nodes. However, with a maximum count of 2 in this sub-graph, triangular relationships appear sparse. This suggests a lower level of clustering compared to denser graphs where nodes participate in multiple triangles. This parallels the findings of the connected-component analysis, highlighting the presence of several connected components instead of a single densely connected component observed in the original graph.

This observation is consistent with the average clustering coefficient of around 0.011, indicating that, on average, nodes in the sub-graph are not densely clustered.

5.2.3 PageRank. By arranging nodes in descending order based on their PageRank scores as shown in Figure 6, we can identify the top influencers in the network. These influential nodes play a crucial role in directing the flow of information and connections within the graph.

In this sub-graph, the user with ID 17166447 holds the highest PageRank score, indicating significant influence within the network.

count	id	Features
2	43003845	[schzimmydeanie, ...]
2	216843160	[schzimmydeanie, ...]
1	8088112	[jony2k, mindywhi...]
1	24117694	[ct_legacy, callo...]
1	12611642	[spacex, nutzareu...]
1	259842341	[jony2k, hayleywa...]
1	21681252	[dj, djxyanyde, a...]
1	19563357	[sarahdope, carbo...]
1	49253437	[andykipling, win...]
1	34428380	[schzimmydeanie, ...]
1	56860418	[paramoreans, jer...]
1	52405864	[gameinformer, sa...]
1	86221475	[jony2k, hayleywa...]
1	93905958	[callofduty, trey...]
1	110315478	[ms_laser, bbh_st...]
1	100318079	[acesso_mtv, live...]
1	160237722	[jony2k, todopara...]
1	104352067	[aqueenofhearts, ...]
1	173732041	[jony2k, liveforh...]
1	106607901	[, opticdi3sel, j...]

Figure 5: Triangle Counting Analysis on the Sub-graph

id	Features	pagerank
17166447	[erichalvorsen, t...]	9.964891800495328
14691709	[13, ctz, callofd...]	8.023647319083862
18951737	[fbrstreetteam, f...]	7.042305156157047
60968363	[aqueenofhearts, ...]	6.4953131903848655
90561258	[mupinbeatjunkie,...]	5.488764858850935
14824849	[alecbaldwin, leo...]	5.3438551162430254
43003845	[schzimmydeanie, ...]	5.079188442190626
43933017	[aqueenofhearts, ...]	5.058187772649208
430268163	[todoparawhore, w...]	5.008570812668531
14270527	[halo, crixlee, p...]	4.997885900477347

Figure 6: PageRank Scores of Top 10 Users in Sub-graph

Following is the user with ID 14691709, boasting a PageRank score of approximately 8.02, and so forth.

5.2.4 Community Detection. We used the 'labelPropagation' function of GraphFrames to discover communities [3] within the sub-graph. Label propagation leverages local information exchange among neighboring nodes, where each node updates its label based on the majority label among its neighbors. This process reveals communities where nodes share similar labels. As shown in the table in Figure 7, several communities emerge, each comprising approximately 10 to 20 nodes. This observation aligns with our earlier analysis, indicating that the sub-graph lacks dense connectivity, characterized by multiple connected components and a low average clustering coefficient.

The user IDs of each community are also available in the provided Colab notebook for more detailed information.

5.2.5 Link Prediction. Predict the likelihood of the existence of a link between two nodes in the sub-graph network [1].

1. Generating a Dataset File for Link Prediction Supervised Learning

label	count
206923844	18
24117694	18
100318079	15
256497288	15
43003845	11
270449528	11
18665800	10
198941747	10
16616109	9
17658786	9

Figure 7: Community detection in the sub-graph

To construct a dataset for supervised machine learning from the sub-graph, we cross-joined the vertices data frame with itself, producing all possible pairs of vertices except those with identical node IDs. Next, we merged the features of each pair into a unified feature vector for subsequent use. Additionally, we verified the existence of an edge between each pair of vertices, assigning a label of 1 for existing edges and 0 for absent edges.

However, the resulting data frame contained an imbalance in label distribution, with a significant surplus of rows labeled as 0. To ensure balanced classes, we sampled an equal number of data instances with label 0 and label 1, resulting in approximately 5838 and 5744 samples, respectively. Finally, we saved the resulting data frame as a CSV file to avoid reconstructing the dataset from scratch, a process that consumed roughly 20 minutes. We made the dataset downloadable with URL in our notebook.

2. Training a Word2Vec Model for Feature Embedding

First, we tokenized the combined features of each pair of vertices connected by an edge. Then, we applied a Word2Vec model to convert these tokenized features into numerical vectors. The trained Word2Vec model was then saved to avoid the need for retraining, which can take up to 20 minutes. Access to the trained model is provided via the URL in our notebook.

3. Partitioning the Dataset into Training and Testing Sets

We partitioned the dataset into training and testing sets using an 80:20 ratio. To ensure reproductibility, we set a seed during the split.

4. Training a RandomForest Classifier for Link Prediction

We employed the RandomForest classifier for binary classification, distinguishing between the presence and absence of an edge between two vertices based on their combined features. To maximize efficiency, we parallelized both the training and testing processes:

1. *Training with Cross-Validation*: Leveraging Spark's 'CrossValidator', we parallelized model training. During cross-validation, the dataset was divided into multiple folds, and the model was trained independently on each fold. This parallel execution across folds optimized the training process.

2. *Testing and Evaluation*: Predictions on the testing data were made in parallel using the 'transform' function, while evaluation with the 'BinaryClassificationEvaluator' occurred concurrently. Spark's inherent parallelization of data frame transformations ensured efficient processing across multiple executors.

Metric	Value
Accuracy	0.860134
AUC	0.929763
Precision	0.838248
Recall	0.890779
F1 Score	0.863715

Figure 8: Community detection in the sub-graph

The results presented in Figure 8 showcase the model's good predictive capability in identifying potential edges within the graph. However, it's worth noting that there's room for improvement, especially through further training of the Word2Vec model. By fine-tuning the Word2Vec embeddings and experimenting with various hyperparameters, we can elevate the predictive capability of the link prediction model.

6 EVALUATION

Evaluation of Correctness and Speed: Distributed vs. Non-Distributed Implementations

We utilized 'sc.parallelize()' to distribute files for parallel reading, leveraging PySpark's distributed capabilities. Despite the small size of individual files (ranging from a few KB to less than 100KB), the total volume of data (265MB) necessitated the use of a distributed approach. To evaluate our PySpark implementations for reading file contents, we explored both distributed (with PySpark) and non-distributed (with simple Python) approaches. As expected, both the distributed and non-distributed methods produced identical results, though there were minor variations in the order of data when collected using PySpark. Notably, the outputs from GraphFrames were consistent across both methods. However, there was a significant difference in runtime: the non-distributed method completed reading RDDs in about 1 minute, while the distributed approach took approximately 24 minutes to process all files (see Figures 8 and 9 in the Appendix for runtime details). This was expected, as the Spark implementation is inherently slower due to framework overhead, but it benefits from parallel execution, making it more scalable. Overall, both implementations met our expectations in terms of yielding the same results, despite the slowdown in the PySpark implementations.

Evaluation of GraphFrames Functionalities

GraphFrames' built-in functions are designed for distributed processing and operate in parallel, utilizing Apache Spark's distributed computing framework to efficiently manage large-scale graph data. Simple operations, like computing node degrees, are executed quickly due to their straightforward nature, which minimizes communication and computation overhead. However, more complex algorithms such as PageRank or Label Propagation require more computational resources due to their iterative processes and increased communication between distributed nodes. As a result, these advanced algorithms perform slower, especially on larger and denser graphs like the one in our project. To balance computational efficiency and analytical depth in exploring Twitter social circles, we extracted a sub-graph to focus on these advanced GraphFrames algorithms. All GraphFrames functionalities provided consistent insights for the entire graph and the sub-graph, enhancing our understanding of the graphs.

Evaluation of Supervised Machine Learning (Link Prediction)

Regarding the Link Prediction task, we employed Supervised Learning and evaluated the model using five metrics: Area Under Curve (AUC), Accuracy, Precision, Recall, and F1 Score. These metrics collectively demonstrated the model's good predictive ability in identifying potential edges within the graph. However, further improvements could be achieved with additional computational resources.

Overall, we successfully accomplished our objectives of leveraging GraphFrames to analyze graph data, applying concepts learned in class such as file reading, relational database operations, and machine learning techniques. Through this process, we have gained valuable skills in analyzing large-scale network datasets from various perspectives, enhancing our capabilities in graph analytics.

7 CONCLUSION

In conclusion, our exploration of GraphFrames has broadened our understanding of distributed graph analytics. We successfully implemented parallel file reading techniques, extending our capabilities beyond what we learned in class. Through GraphFrames, we delved into fundamental graph operations and advanced algorithms like PageRank and label propagation, gaining insights into complex network structures.

Working with GraphFrames has expanded our knowledge of graph operations and improved our ability to use relational databases effectively. We've also developed skills in advanced methods such as parallel k-fold cross-validation and testing, making our model evaluations more efficient. Additionally, we learned how to embed textual data into numerical vectors using Word2Vec models.

The primary challenge we faced throughout the project was the constrained computational resources within the Google Colab environment. Given the substantial size of our dataset, attempting to execute advanced GraphFrames algorithms and machine learning models on the entire graph led to program crashes in Colab. As a solution, we decided to work with a sub-graph to manage these resource limitations and achieve smoother execution. If additional computational resources become available in the future, we aim to explore the feasibility of applying advanced algorithms to the entire graph.

REFERENCES

[1] U. Alfaiz, M. Subhadhriti, D. Aritra, and S. Prince. 2018. Social Networks Analysis : Link Prediction. RCC Institute of Information Technology. (2018). https://rcciit.org/students_projects/projects/cse/2018/GR16.pdf Accessed: 2024-04-08.

[2] M. Julian and L. Jure. 2012. Social circles: Twitter. Stanford Network Analysis Project (SNAP). (2012). https://snap.stanford.edu/data/ego-Twitter.html Accessed: 2024-04-08.

[3] G. Kedar, R. Raksha, S. Jason, and D. Dana. 2023. Parallelized Community Detection in Social Networks. GitHub. (2023). https://github.com/kedarghule/Community-Detection-in-Social-Networks Accessed: 2024-04-08.

[4] GraphFrames Development Team. 2023. *GraphFrames User Guide*. https://docs.databricks.com/en/integrations/graphframes/user-guide-python.html Accessed: 2024-04-08.

[5] H. Timothy and B. Joseph. 2016. GraphFrames Python API docs. (2016). https://graphframes.github.io/graphframes/docs/_site/api/python/index.html Accessed: 2024-04-08.

A APPENDIX

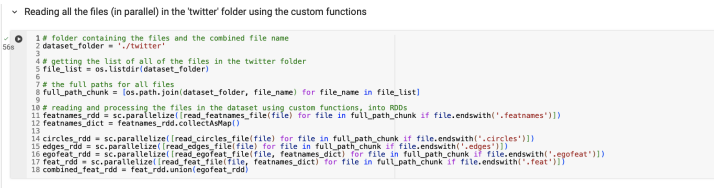

Figure 9: Running Non-Distributed Implementation of File Reading Custom Functions (With Simple Python)

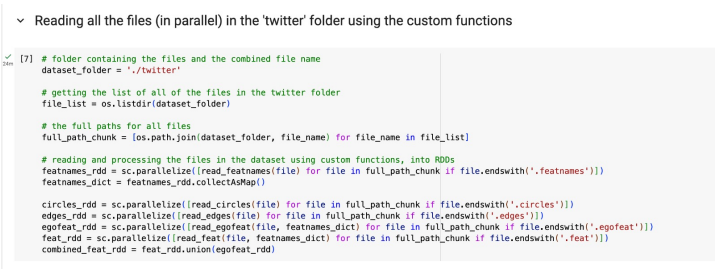

Figure 10: Running Distributed Implementation of File Reading Custom Functions (With PySpark)