SOCIAL NETWORK ANALYSIS AT SCALE: GRAPH-BASED ANALYSIS OF TWITTER TRENDS AND COMMUNITIES

by

Lia Nogueira de Moura, B.A.

A thesis submitted to the Graduate College of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Computer Science
December 2020

Committee Members:

Jelena Tešić, Chair

Vangelis Metsis

Adrienne Hall-Phillips

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

DEDICATION

To my wonderful husband, Jeremy Trahan, who has supported me every step of the way. From helping me brainstorm ideas and proofread my writing, to being patient with my frustrations along the way and helping me with my work/school/life balance. Thank you for supporting me in the tough moments, and enthusiastically celebrating my accomplishments as if they were your own.

ACKNOWLEDGEMENTS

Thank you to my amazing advisor, Dr. Jelena Tešić, who has guided and inspired me throughout my graduate studies from the very beginning. Thank you for being supportive, responsive, flexible with my schedule, and patient with my limitations. I am so grateful to have women in computer science that I can look up to.

Thank you to Mirna Elizondo, a positive, driven, and hard-working undergraduate student at Texas State University, for her assistance with running experiments and testing my code.

I would also like to thank Dr. Hall-Phillips and Dr. Metsis, for participating in my defense committee.

A special thanks to Dr. Hall-Phillips for contributing one of the datasets used in this thesis (*MeToo* dataset).

And last, an additional thanks to my husband, Jeremy Trahan, for helping with proofreading my writing.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

ABSTRACT

Twitter's influence on society and communication has motivated research work in the past decade. A large percentage of existing research focuses on specific Twitter datasets bound by time, location, topic, hashtag, and the analysis of the content of tweet messages of said datasets, and their influence on the fields of business, education, geography, health, linguistics, social sciences, and public governing. Researchers have attempted to answer a variety of questions, e.g. "What topics are being discussed in the Twitter dataset?", "What communities are formed within the set of users?", "Which users are at the center of a particular discussion?", "How are users reacting to real-time events?", and more important, "How can we combine and refine existing data science techniques that can be used in other Twitter research related work?" There have been very few attempts to address the scale and design of an end-to-end data processing and analysis pipeline at scale. This body of work offers one solution for a scalable way to gather, discover, analyze, and summarize joint sentiment of Twitter trends (topics, hashtags), and communities (groups of users that are bound by connection, topic, time period, or possibly location/language/interest) in the larger subspace of the Twitterverse. Topic discovery is improved by contextual network construction and tweet aggregation. The work offers an overarching pathway on how to construct an end-to-end data science pipeline for meaningful analysis of Twitter datasets at scale, namely data management, graph network construction, clustering, topic modeling, and graph data compression for meaningful visualization. We evaluate the data science package and different methods for graph construction and tweet data processing on over 12 million tweets over six different Twitter datasets.

I. INTRODUCTION

Consumers lead rich digital lives, and demand real-time personalized service and delivery of content. Brands and companies are continuously calibrating their strategies to win over informed and impatient connected consumers by expanding their portfolio to include more on-demand and personalized products and services. Companies and governments share official announcements, handle customer service, issue clarifications, and use Twitter trends to promote the brand. Social media has increasingly taken a bigger role in shaping public opinion and public policies in the past decade. The Twitter platform provides a fast condensed exchange of information, opinions, and data, as Twitter's discussions and topic trends are increasingly responsible for shaping campaigns, policy issues, and marketing strategies for businesses and governments. Social media users express their opinions, communicate with each other, discuss topics, spread news, and influence each other all over the globe. The large number of users, and even larger number of messages shared, commentated, and propagated create interesting data, and bring important insights about human behavior, marketing, linguistics, industry trends, brand monitoring, politics, etc. How do we separate the useful information from the millions of content items the consumer shares online from the background noise, or irrelevant trolling? Here, we propose to scale social data analysis, and allow the user to distinguish the relevant content and guide the inquiry. We introduce a scalable way to gather, discover, analyze, and summarize joint sentiment of the Twitter communities. The contribution of the thesis is (i) a software package for general use [1] that implements a data science processing pipeline for Twitter data, and (ii) a suite of graph-clustering and topic modeling techniques improved for large datasets, and robust content analysis of noisy communities on Twitter [2].

The thesis is organized as follows: Chapter II summarizes state-of-the-art

efforts in social network analysis, graph clustering, topic modeling, and Twitter data processing at scale. Chapter III covers graph network construction, introduces methods for multi-modal network construction where an edge in the graph can be constructed from a variety of connection modes (e.g. retweets, replies, quotes, mentions, and hashtag). We implement a set of graph-based community detection approaches and propose methods for community discovery at scale. We introduce a suite of visualization and graph reduction techniques to help interpret the meaning of the resulting communities. Chapter IV focuses on topic discovery in Twitter communities. Here, we address the semi-structured nature of tweets e.g. short, slang-, acronym-, and symbol- rich modes of expression, bursting with misspellings, icons, and symbols. Chapter V focuses on the data science aspects of dealing with semi-structured tweet data at scale, and introduces specific data management (cleaning, ingestion, interpretation, and storage) techniques for improved performance. Chapter VI summarizes the documentation of the released software package *pytwanalysis* [1], namely outlining how to reproduce the work presented in the thesis: tweet extraction, cleaning, storing of tweets in MongoDB, graph-clustering, topic discovery, and visualization tasks. Chapter VII contains experiments and results when the proposed system is used for analysis of six different Twitter databases of over 12 million tweets combined. The thesis concludes with Chapter VIII.

II. RELATED WORK

With the rise of Twitter social media platform influence, data science researchers started to take notice. Williamns et al. (2013) [3] collected over 1000 papers related to microblogging and Twitter published between 2007 and 2011 to classify the emerging research trends on tweet data analysis, and classified the work along four dimensions: message, user, technology, and concept. The bulk of the research is focused on the message and the user, i.e. content of the tweets and communication trends among users on the platform. Bruns et al. (2014) [4] expands the survey and confirms that most of the research focus is on content analysis of the tweets. Research work to date has used trends in Twitter for tracking the influenza [5], detecting communities and opinions about climate change [6], and using Twitter mood to predict stock market changes [7]. Twitter data have also been used to understand the influence of fake news in Twitter during the 2016 US presidential election [8], analyze the COVID-19 and the 5G Conspiracy Theory [9], and the COVID-19 Twitter narrative among U.S. governors and cabinet executives [10]. Twitter communication trends, generated content, and interactions influenced a variety of aspects of social life, and researchers focus on community analysis, and topic discovery to model public opinion in geography [11], health [12], linguistics [13], political sciences [14], economics [15], government policy [16], and political consumerism [17].

The study of social networks is an essential component in the understanding of the fake news phenomenon. Of particular interest is the network connectivity between participants, since it makes communication patterns visible. These patterns are hidden in the offline world, but they have a profound impact on the spread of ideas, opinions and news. Among the major social networks, Twitter is of special interest, as it offers the possibility to perform research without the risk of breaching

the expectation of privacy.

A variety of graph clustering algorithms identify different structures within a network with the focus of how densely connected the vertices are [18, 19] or discovering two opposing communities [20, 21]. The definition of communities on Twitter is multi-dimensional, as users in the community can share one or more commonalities. Communities can be formed based on: how similar the users are, how close geographically they are to each other, how connected the users are to each other by friendship, by replies and mentions, or communities of people talking about the same subject [22]. The lack of ground-truth makes the evaluation challenging, as it is difficult to know for sure if the groupings are correct [23]. The *Latent Dirichlet Allocation (LDA)* topic modeling tool, from text analysis, gains its prominence in social media topic discovery [24, 25]. Since tweet records are a relatively new form of communication, current research focus is on improving and extending the pre-processing of content, and the network of tweets to help LDA achieve better results [26, 27].

Current tools used for Twitter analysis focus on the specific project, and fail to generalize the end-to-end process. Tools such as Gephi [28] and Pajek [29] are useful for visualizing networks and can be easily used by someone without programming experience. Other tools, such as NodeXL [30], offer more than just visualizations, and can extract data using Twitter API, run graph analysis, and create visualizations, but it is limited to only Windows operating system and small subsets of data. There are also more complex and comprehensive tools for network analysis that are built for the more advanced user, such as the SNAP library [31].

Twitter content is semi-structured in its origin: while users and words are consistent, content, use of hashtags, location information, and connections are dynamic, noisy, and inconsistent. Different SQL data warehouse approaches have been suggested for organizing Twitter data [32, 33, 34, 35] to make it fit into the

structured data mold. It works for a small subspace and specific tasks of the Twitterverse, but it fails to generalize to scalable and subject agnostic systems. Document driven solutions support the semi-structured and ever evolving nature of tweet records, and scientists have moved to NoSQL solutions to support data management for their projects. TweetCuriosity [36] retrieves tweets with given user parameters, cleans retweets and location statements, stores clean tweets into NoSQL MongoDB, and analyzes and visualizes the results on a quarter of a million tweets. Saini et al. [37] proposed enrichment of electronic healthcare records (EHR) data analysis using Twitter data source: MongoDB maps tweets to electronic healthcare records, and enhances the medical domain data mining. Social media search app Twoogle [38], built on top of Baqend's real-time query API, allows searching Twitter with MongoDB queries.

Researchers have proposed a data science pipeline for a limited dataset for the specific application of integrating machine learning modules for fake news identification [39]. While they achieve scalability for the Twitter dataset analysis by parallelizing the process, the dataset and specific analysis (fake news) limit the extension and generalization of the pipeline [39].

There has been no known attempt to create a scalable unified framework for end-to-end gathering, processing, storing, multi-view analysis, community discovery, topic analysis, and visualization tools that can support different directions of Twitter research data at scale. In this thesis, we propose a scalable way to gather, discover, analyze, and summarize the joint sentiment of the communities online, that can be used in a variety of other future studies which, like many others, are interested in understanding the content of the messages of a given Twitter dataset. The data management part resulted in the data science analysis package *pytwanalysis* [1] which can handle a variety of data science projects using Twitter data at scale, and allow a high level of customization of the pipeline and MongoDB data storage. The

data analysis step improved upon existing graph-clustering and topic aggregation methods to study the content of a Twitter dataset. The proposed approach takes a fresh look at different topics being discussed within a Twitter dataset as it creates multiple overlapping partitions in the data based on different criteria, but driven by the same goal of grouping discussions together. We present a scalable framework for gathering the graph structure of follower networks, posts and profiles, and show how to use the collected data for high-performance social network analysis.

III. GRAPH ANALYSIS

Social network analysis (SNA) investigates social structures through the use of networks and graph theory. Graphs contain vertices (nodes) and edges (connections); in SNA, vertices can be users, groups of users, and communities. The vertices in the network can be tied together by specific types of relations, such as friendships, belonging to an organization together, or even having similar interests. These relationships are the edges of the network. Studying such structures may help us understand how the network is organized, and may lead to the discovery of patterns among the participants, communities, and possibly to new insights in answering questions for a variety of disciplines. From analyzing the social behavior of criminal networks as an investigative tool [40], to analyzing the potential applications for logistics and supply chain management research [41], or understanding American politics [42].

In social media services, such as Twitter, users can connect in many different ways: following other users, re-tweeting or liking the content of other users, and replying and/or mentioning users in discussion threads. Hashtag (#) labels of social media content are a streamlined way to "attach" a user's content to a trend and/or topic. A recent study [43] treats hashtags as a functional means to structure content, and outlines evidence that the role of hashtags in social media content expands from structuring (spreading) content to an integral role of contemporary communication in the media. The study presents a series of six empirical studies to systematically assess motivations for using hashtags, and uncovers the existence of roughly 10 categories of hashtag use: entertainment, content organization, design, engaging in trend topic, bonding over social cause/event, inspiring social or policy response in real life, reaching a wider audience, summarizing the content, and endorsing a specific product, cause, lifestyle, etc. The authors demonstrate how

these motivations differ between platforms, and how they relate to different patterns of social media behavior [43]. Twitter trending hashtags have changed the way we consume and actively search for the relevant news in all domains. Social impact theory on the consumption practices of individuals discovers an important role when applied to hashtag analysis in Twitter data related to political consumerism [17], and hashtag analysis uncovers flavors of political consumerism that change the sentiment of the tweet and subsequent actions. Hashtags have become an important means of communication and connection on Twitter. Hashtags influence trends and exploit the meaning of trends through hashtags in social, political, economic, and sociological realms [43, 17].

Each of the connections mentioned (user initiated or hashtag inferenced) and their combination is an opportunity to create a different social analysis network for in-depth analysis. In this chapter, we introduce methods for multi-modal network construction for Twitter data where edges in the graph can be constructed from a variety of connection modes. We implement a set of graph-based community detection approaches and propose a community discovery at scale. We introduce a suite of visualization and graph reduction techniques to help interpret the meaning of the resulting communities. Social network graphs can go from a few vertices to a few million vertices [44]. For large complex graphs, it is hard to see the tree from the forest. Graph summarizing (aggregation, compression, reduction) techniques were proposed to simplify the graphs for better understanding. Proposed work on summarization focuses on techniques that allow for • discovering a high-level description of the influence propagation of users and topics [45], • visualization that is time and processing power bound [45].

Graph Network Construction

Twitter data is provided in the form of Twitter documents, as described in Chapter V. The different Twitter APIs have different information available. Since the focus of this work is on the content of the tweets, we use the APIs that provide the tweet content with additional information about each tweet [46, 47, 48]. From the documents retrieved from the APIs, it is possible to know the text of the tweets, the hashtags used, the user that created each tweet, the users that were mentioned, as well as the user of the original tweet in case of retweets, quotes, and replies. Even though each record does provide a count of how many times that tweet was liked, and how many friends and followers each user has, it does not contain the list of all followers, friends, or people that liked the tweet. A separate API that focus on the user account activity [49] is available and returns the information and relationships for a particular user, but separate requests to the account activity API are required to retrieve the additional information for each of the users found from tweet content APIs. Although these requests could be automated, the API's rate limits make this process impractical. And the relationships (follows, friendships, likes) alone don't tells us about the content we want to focus on.

We propose different ways of creating undirected networks based on the information available in each tweet document. The graph terminology used in this chapter is outlined in Table 1. The networks were built by creating users and hashtags as vertices and the following connections as edges:

• The **Retweets Only** network creation method creates a connection between users based on their retweets. Let x and y be two Twitter users. An edge *(x, y)* gets created in G when either x retweets a tweet from y, or y retweets a tweet from x. The *weight(x, y)* will be the total number of retweets from each direction. A self-loop *(x, x)* is created when user x retweets their own tweet. Vertices with a high

Table 1: Graph terminology

Notation	Description
G	An undirected graph G.
G[S]	An induced subgraph of graph G for vertex subset S.
V, n	Vertex $v \in V$ vertex-set; $n = \|V\|$, number of vertices.
E, m	Edge $e, e \in E$ edge-set, $m = \|E\|$, number of edges.
k	Number of clusters.
deg(u)	The degree of vertex u.
weight(u,v)	The weight of edge (u, v).
N[u]	Closed neighborhood of u, a sub-graph induced by all vertices that are adjacent to u.

degree will represent users that were retweeted frequently. The weight of each edge was calculated based on the number of retweets between two users (undirected).

• The **Quotes Only** network creation method creates a connection between users based on their quotes. Let x and y be two Twitter users. An edge *(x, y)* gets created in G when either x quotes a tweet from y, or y quotes a tweet from x. The *weight(x, y)* will be the total number of quotes in each direction. A self-loop *(x, x)* is created when user x quotes themselves. The main difference between a retweet and a quote is that in a quote more content is added to the tweet. It usually expands on the content of the original tweet. The weight of each edge was calculated based on the number of quotes between two users (undirected).

• The **Replies Only** network creation method creates a connection between users based on their replies. Let x and y be two Twitter users. An edge *(x, y)* gets created in G when either x replies to a tweet from y, or y replies to a tweet from x. The *weight(x, y)* will be the total number of replies in each direction. A self-loop *(x, x)* is created when user x replies to their own tweet. The weight of each edge was calculated based on the number of replies between two users (undirected).

• The **Mentions Only** network creation method creates a connection between users based on their mentions. Let x and y be two Twitter users. An edge *(x, y)* gets

created in G when either x mentions y in a tweet, or y mentions x in a tweet. The *weight(x, y)* will be the total number of mentions in each direction. A self-loop *(x, x)* is created when user x mentions themselves. In the the mentions only network, even if a user y doesn't tweet, retweet, or reply to anybody, they can still show up as a high degree vertex in the mentions network in case thousands of other people mentioned them in their tweets. The weight of each edge was calculated based on the number of mentions between two users (undirected).

• The **All User Connections** network creation method combines all user connections together: retweets, quotes, replies, and mentions. The weight of each edge was calculated based on the number of retweets, quotes, replies, and mentions interactions total.

• The **Hashtag Connections** network creation method identifies hashtags that are frequently used together. Let $\#x$, $\#y$, and $\#z$ be the hashtags used in a single tweet. The hashtag connections network method will create the pairs *{#x, #y}*, *{#x, #z}*, and *{#y, #z}* for this single tweet. Edge weight indicates how many times two hashtags were used together.

Note that we are focusing only on undirected graphs for the simplification of the problem, but the presented analysis could be extended to include directed graphs as well.

Community Discovery in Social Network Graphs

The use of social network analysis to find community structures can give us a better understanding of that network and can give us new insights about how it is organized [50]. Clustering algorithms can be useful for finding such structures, as they focus on the task of grouping members that are close to each other in some way, creating clusters (or groups). Clusters retrieved from social networks are called communities, and the clustering algorithms to find these communities are called

community discovery.

In a social network graph, a community is a subset of vertices within the graph, and the connections between those vertices are discriminant (denser, stronger) with respect to the rest of the network. There are several types of clustering algorithms that can be applied to different tasks. We focus on algorithms that can perform better on network data.

Twitter communities are loosely defined, and can be very different depending on what the research question is: users based on location, message content, hashtag, following, retweets, etc. Context drives both network construction and community discovery [51], and results can vary based on the selection. We adopt the approach of comparing community discovery algorithms based on what the goal of the data analysis is.

Connected Components

A connected component of a network graph is a subset of vertices in the graph such that each pair of vertices is connected by a path. Social networks can have several separate connected components, which would indicate that there is no path of connection among the members of one component to the other, even though they belong to the same network. That information can be helpful when trying to get a first understanding about the network and selecting the correct algorithms for finding communities. For real world graphs and network graphs where connections are guided by a specific subject, we observe very large connected components along with a few very small ones. The largest component can contain over 90% of the vertices, and the rest of the network is divided into a large number of small components disconnected from the rest [44].

Depending on how the network is constructed, there could be different numbers of disjointed graphs of a similar size. The different scenarios require different

community discovery approaches, and to make a correct decision, the first step in exploratory data analysis is to identify the total number of connected components, their sizes, and plot the connected component size histogram. An example of the disconnected graph distribution of a network with a total of 150K vertices is illustrated in Figure 1, where there is one very large connected component, and the second largest connected component has less than 300 vertices. Connected component separation is the first step in the community analysis, and if the largest one has over 90% of the vertices, we focus only on the analysis of that component and group the small ones in the outlier community. If a graph has multiple large connected components, we analyze them separately.

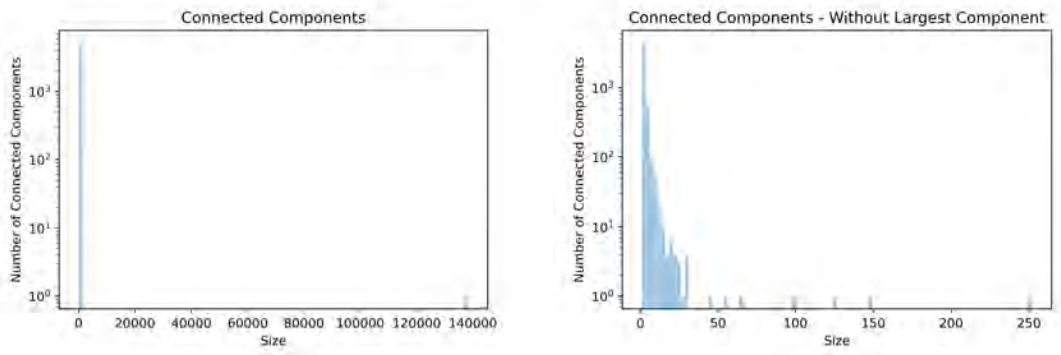

Figure 1: Distribution of the sizes of the connected components for a graph with a total of 150K vertices. The left image shows all connected components, and the right image only excludes the largest connected component.

The method *connected_component_subgraphs* from the networkX package [52] was used to find the list of connected components for the undirected graphs.

Louvain Community Detection

The *Louvain* Community Detection method [53] is a widely used approach to identify communities in large network graphs. It was introduced specifically for the task of finding communities in networks, and it has already been implemented in available libraries that can integrate with existing graph analysis tools such as the

networkX [52] python package. These are the reasons why it was chosen as one of the algorithms used for this analysis.

It is a heuristic method based on modularity optimization that has been shown to outperform other known community detection methods in terms of computation time without losing quality. The *Louvain* method is a greedy optimization method with a runtime that increases close to linearly with the number of vertices in the network, and it has exhibited good summarization power compared to other methods [54]. The outcome of the *Louvain* Community Detection is a set of disjointed communities with densely connected vertices. The weight of the vertices is not considered in the algorithm. Figure 2 shows the visualization of the steps of the algorithm as shown in the original paper [53].

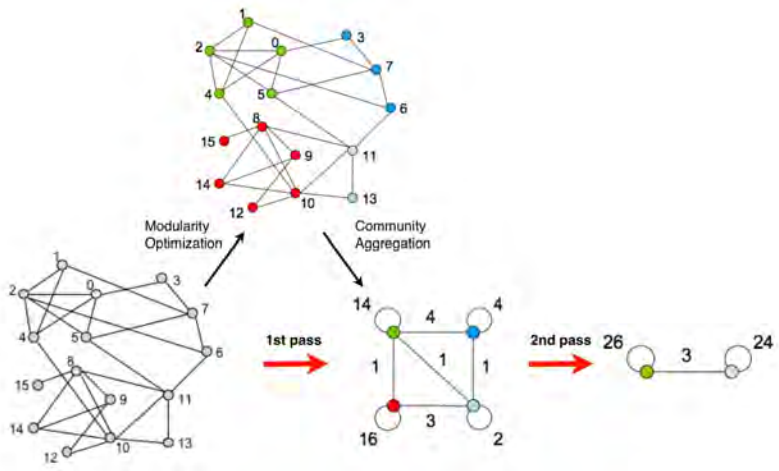

Figure 2: *Louvain* algorithm. In each iteration modularity is optimized for local community changes first, and newfound communities aggregation second. Iterations stop when it is impossible to increase modularity.

Spectral Clustering

Spectral Clustering can be used for cases where the strength of the connection between vertices is important. The *Spectral Clustering* approach performs the *kmeans* clustering method on k eigenvectors corresponding to the smallest

eigenvalues (not 0) of the Laplacian of the adjacency matrix of the graph. Laplacians can be defined in multiple ways, and based on that, there exist many different implementations. *Spectral Clustering* depends on the k, the number of clusters desired, and we select k that maximizes the eigengap [55]. The output is a set of disjointed communities with densely connected vertices. The method has a high computation time and can only be used in fully connected networks, but it uses a similarity matrix to identify how similar the vertices are.

Using the method *SpectralClustering* from the *sklearn* [56] package, each vertex gets assigned to a cluster. To transform the graphs into adjacent matrices that can be sent as parameters to the *SpectralClustering* method, the *to_scipy_sparse_matrix* method from the *networkX* package [52] was used.

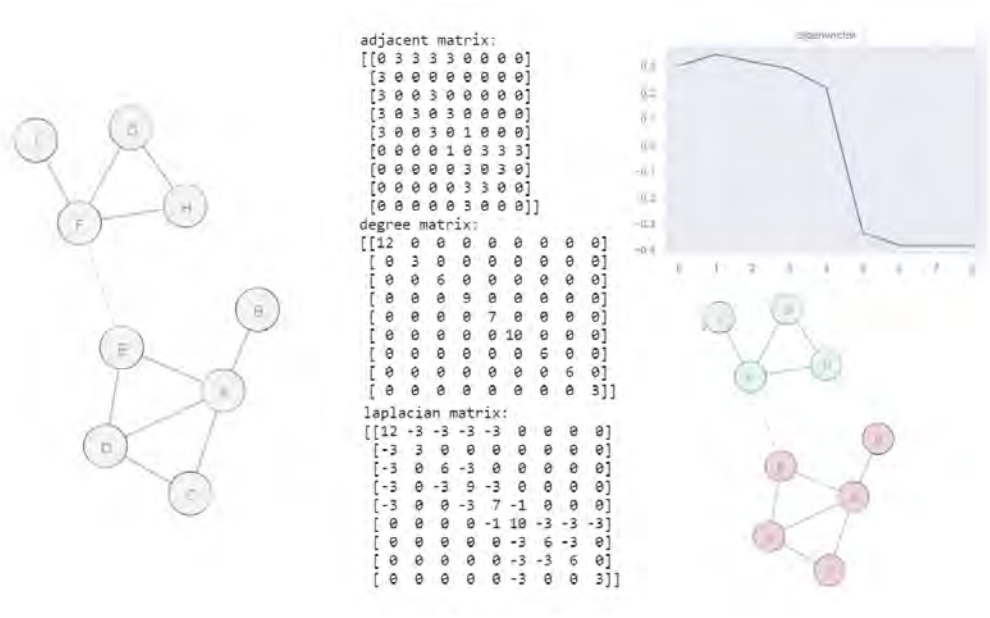

Figure 3: *Spectral Clustering* algorithm. Original graph without clustering (left); the matrices calculation (middle); eigenvector example, and cluster-colored resulting graph (right).

Top k Degree Vertices' Neighborhoods (TN-Neighborhoods)

The majority of tweets come from a small group of *extremely active* Twitter users [57]. According to the Pew Research Center Study, 80% of all tweets come from just 10% of all U.S. Twitter users. These super users tweet 138 times per month, while the median Twitter user tweets twice per month [57]. There is a great need for understanding how these super users shape Twitter communities and guide topic discovery and development. In addition, one super user can be active in multiple communities, based on the tweet content, and can be connected to various users depending on the overlapping commonalities. Here, we propose a novel approach for building communities around super users with relaxation criteria that a user can belong to multiple communities. Graph-theoretic relaxation of the concept of cluster graphs introduced overlaps between the clusters [58], and an adaptive K-means algorithm was proposed for overlapped graph clustering [59]. We propose the cluster overlap approach based on the most influential users in the dataset. We measure the influence of vertex v by its degree in the graph. For a network graph G, and a vertex v, $v \in V$, create a neighborhood sub-graph $N(v)$. The sub-graph will contain all vertices that are connected to v and all the edges that connect the vertices in the sub-graph together. We grow these communities around the highest degree vertices in the dataset. Since users can be connected to multiple high degree vertices, the result of this method is a set of overlapping communities, and the communities capture discussions initiated or related to a set of high influence vertices v.

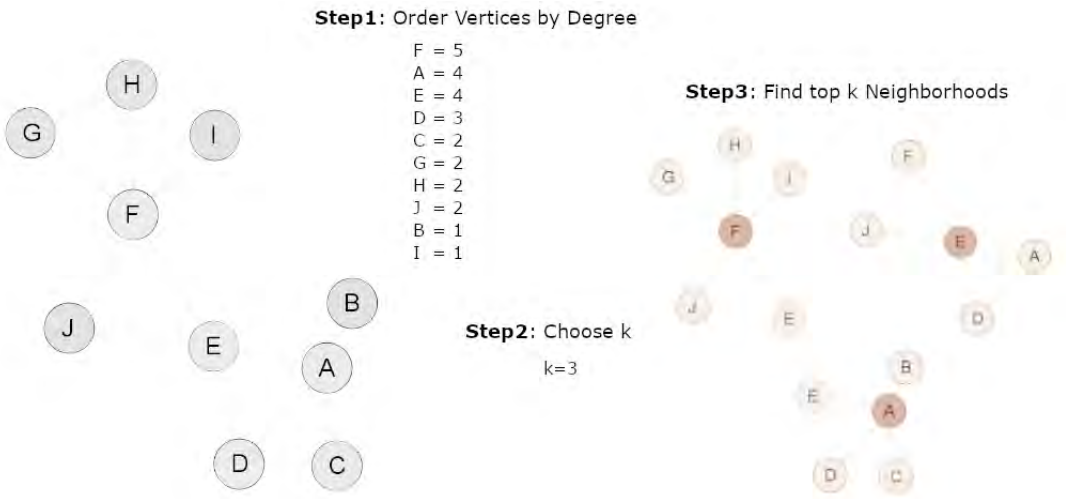

Figure 4: *TN-Neighborhoods* algorithm. Visualization of the high level steps of the method.

Performance Baseline

In order to evaluate the community discovery methods, manually annotated ground truth data were used to establish a baseline. Ground truth data can be used as an ideal expected outcome, and can be helpful for observing the behaviour of real communities.

Graph characteristics for each dataset were recorded and compared. *Separability*, *Density*, and *Clustering Coefficient* were used as the goodness metrics [23], and *Power Nodes Score* metric was introduced to score the communities on how well connected the top degree vertices are. The average degree of vertices, graph clique size, and number of cliques were also recorded for comparison.

Measures of Performance

Separability measures the ratio between the edges of the community (internal vertices) and the edges of the vertices of the community that are pointing to the outside, under the assumption that solid communities would be well-separated from the rest of the network. Let $G[V]$ be the induced sub-graph of graph G for vertex

subset V, where V was created from one of the clustering methods. And let E_m be the number of edges in *G[V]*, and E_c the number of edges on the boundary of *G[V]*. Then we calculate separability as $sep = \frac{E_m}{E_c}$. **Density** measures how well connected the vertices are. Let *G[V]* be the induced sub-graph of graph G for vertex subset V, where V was created from one of the clustering methods. And let E_m be the number of edges in *G[V]*, and V_n the number of vertices in V. Then we calculate density as $den = \frac{2*E_m}{V_n*(V_n-1)}$. **Power Nodes Score** identifies graphs that are highly connected through a set of highly connected vertices. A high score will indicate that the top degree vertices of the graph are highly connected to all other vertices in the graph. Let *G[V]* be the induced sub-graph of graph G for vertex subset V, where V was created using the *TN-Neighborhoods* approach for the highest-degree k vertices. If V_n is the number of vertices in V, and n the number of vertices in the original graph G, the power vertices score is calculated as $pns = \frac{V_n}{n}$. Note that k is the number of clusters for all clustering methods, and we use the same k in all methods for comparison. **Average Clustering Coefficient** is the average of the local clustering measures for each vertex. The local clustering of each vertex in G is the ratio between the triangles that actually exist and all possible triangles in its neighborhood [60]. It was proposed as a goodness metric on the premise that pairs of vertices with the same neighbors are likely to be connected to each other. **Average Degree of Vertices** quantifies the average degree of all vertices in G, and measures graph connectivity. Let V be the vertex-set of G, and n the number of vertices in V. Then calculate the average degree of vertices as $av_n = \frac{\sum_{i=1}^{n} V_i = deg(V_i)}{n}$. **Graph Clique Size** measures the number of vertices in the largest clique of G. A clique C in an undirected graph G, is the induced sub-graph of G where every two distinct vertices in C are adjacent [61]. In social networks, cliques could represent groups of people where everyone in the group has a connection with every other person in that same group. **Number of Cliques** is the number of maximal cliques in G.

Datasets

Datasets with manually-curated ground truth communities are used as a baseline [62]: • **football:** A dataset with football players and clubs from the English Premier League. The data contains 248 Twitter users grouped into 20 clubs in the league. Each user belongs to only one club; • **olympics:** A dataset with athletes and organisations that were part of the London 2012 Summer Olympics. The data contains 464 Twitter users grouped into 28 different sports. Each user belongs to only one sport; • **politics-ie:** A dataset with Irish politicians and political organisations. The data contains 348 Twitter users grouped into 7 different groups based on the user's affiliation. Each user belongs to only one affiliation; • **politics-uk:** A dataset with Members of Parliament (MPs) in the United Kingdom. The data contains 419 Twitter users grouped into 5 different political parties; and **rugby:** A dataset with international Rugby Union players. The data contains 854 Twitter users grouped into 15 overlapping countries;

Table 2: Number of users and communities available for each of the ground truth datasets

Dataset	# of users	# of communities
football	248	20
olympics	464	28
politics-ie	348	7
politics-uk	419	5
rugby	854	15

Baseline Findings

For each of the 5 datasets [62], the *Louvain*, *Spectral Clustering*, and *TN-Neighborhoods* methods were run. Their performance was measured using clustering measures and ground truth data. Table 3 summarizes findings for all 5 datasets, ground truth data analysis, and clustering methods.

Table 3: Baseline data. db is the dataset name, n is the number of vertices in the graph, k is the number of communities, ave_deg is the average degree of vertices across communities, sep is separability measure, den is density measure, acc is the average clustering coefficient, pns is the power nodes score for the top $k = 3$ users, gcs is the graph clique size, and nc is the number of cliques.

Ground truth data									
db	k	n	ave_deg	sep	den	acc	pns	gcs	nc
football	20	248	9.313	0.324	0.188	0.886	0.994	8.800	6.600
olympics	28	464	11.752	0.285	0.210	0.838	0.990	10.500	36.857
politics-ie	7	348	28.294	0.492	0.119	0.840	0.986	18.714	17150.571
politics-uk	5	419	45.482	0.937	0.154	0.729	0.961	26.600	6141372.6
rugby	15	854	21.039	0.361	0.334	0.464	0.800	13.200	2094.800
Louvain Clustering									
DB	k	n	ave_deg	sep	den	acc	pns	gcs	nc
football	13	248	10.076	0.359	0.131	0.811	0.936	9.692	15.231
olympics	9	464	17.907	0.496	0.062	0.715	0.870	14.889	318.778
politics-ie	5	348	37.740	0.674	0.041	0.777	0.971	23.800	24175.800
politics-uk	3	419	71.563	1.525	0.020	0.777	0.973	41.667	5950766
rugby	9	854	35.330	0.636	0.026	0.690	0.914	22.333	2529.444
Spectral Clustering									
DB	k	n	ave_deg	sep	den	acc	pns	gcs	nc
football	20	248	7.899	0.287	0.330	0.811	0.991	7.500	13.450
olympics	28	464	9.201	0.315	0.466	0.710	0.989	8.036	240.464
politics-ie	7	348	25.800	0.487	0.400	0.717	0.996	17.286	21740.143
politics-uk	5	419	43.723	0.969	0.548	0.483	0.944	25.400	6164034.200
rugby	15	854	25.437	0.442	0.080	0.798	0.972	17.133	3215.133
TN-Neighborhoods									
DB	k	n	ave_deg	sep	den	acc	pns	gcs	nc
football	20	248	17.146	0.483	0.032	0.602	1.000	10.450	280.750
olympics	28	464	35.632	0.640	0.015	0.584	1.000	18.929	4091.071
politics-ie	7	348	77.600	2.207	0.009	0.643	1.000	35.143	242155.571
politics-uk	5	419	104.169	2.259	0.007	0.672	1.000	58.400	15649452.400
rugby	15	854	43.199	0.480	0.009	0.518	1.000	21.267	11138.067

Figure 5 visualizes the comparison between the average goodness metrics of all the communities found using the different clustering methods, executed on the different datasets.

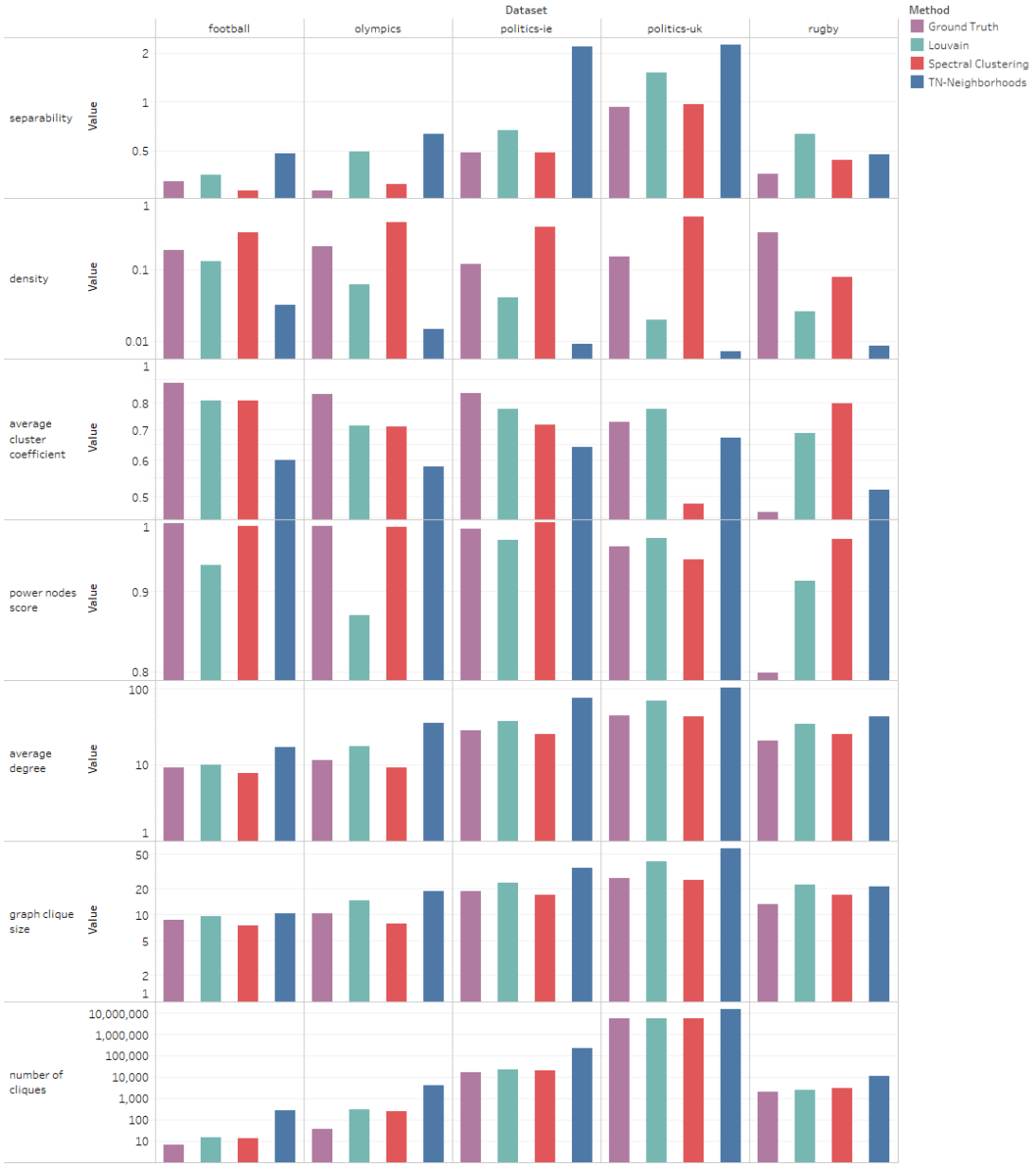

Figure 5: The average metrics of all the communities found using the ground truth communities and the different clustering methods executed on the different datasets

Focusing first on the ground truth data that were taken from real communities, we can observe that each dataset performs differently among the different metrics as

shown in Figure 5. The *politics-uk* dataset has higher *Separability* than all other datasets, while the *olympics* dataset has the lowest with a very low score. The *football* dataset has a very high *Average Clustering Coefficient* and a very high *Power Nodes Score*, while the *rugby* dataset has very low score for the same metric. Even though these communities are real, they do not necessarily receive a high score for the goodness metrics, which could indicate that the metrics, rather than indicating whether a community is good or not, give us an intuition about the characteristic differences among the different datasets. The *politics-uk* dataset appears to have the most connected and well separated communities and have a large number of groups within the communities that are all connected to each other. The *rugby* dataset on the other hand, even though it is very dense, it has a very low *Average Clustering Coefficient* and *Power Nodes Score*, which could infer that there are not many pairs of members in the communities with the same neighbors.

Comparing the ground truth metrics with the metrics from the clustering algorithms, we can observe that the *TN-Neighborhoods* method has the highest scores for *Separability* and *Power Nodes Score*, and the the largest clique sizes and number of cliques, but the lowest *Density*. Even though the *TN-Neighborhoods* method performed well in many of the metrics, it was also the method with the largest distance from the ground truth metrics, which could indicate that it was the method that created communities that are the farthest apart from the manually-annotated communities. The *Spectral Clustering* method found communities that are very dense, and the *Louvain* method communities that are better separated than the *Spectral Clustering*. Both the *Spectral Clustering* and *Louvain* method metrics are closer to the metrics in the ground truth data than the *TN-Neighborhoods* method, which could indicate that they were able to find communities that are closer to the real communities. Analyzed clustering algorithm performance varies depending on the dataset, as communities in the different

datasets contain different characteristics. Experiments to measure the detailed differences between the algorithms on multiple datasets are presented in Chapter VII.

Data Aggregation and Visualizations

Large networks can contain millions of vertices that connect to each other in some way. Community discovery allows us to analyze one community at a time. When a community contains tens of thousands of vertices, visualizing the network as an entity does not provide a lot of information, as shown in Figure 6.

Data aggregation and grouping of vertices enable meaningful community analysis. The specific technique employed depends on the research question we are trying to answer, what will have a low cost if discarded, and what the aggregation is meant to accomplish, as they can focus on graph aggregation to maintain the same structure, identify patterns, maintain the most salient information of the original graph, or focus on the most influential vertices. Various graph aggregation approaches have succeeded in reducing the graphs without major structural changes [54, 63], while reducing the high processing cost.

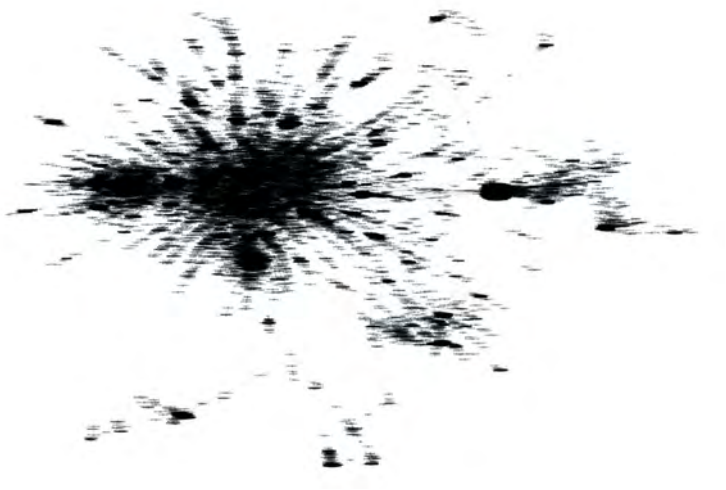

Figure 6: Network example with no reduction. Graph with 34K vertices and 41K edges.

We propose and implement context-aware graph reduction methods that allow for fast and meaningful visualization: • **Community Percentage:** A percentage of vertices of every community found in a given graph is removed, starting from the lower degree vertices. This approach reduces the number of vertices to plot, without changing the overall structure of the data. The *Louvain* method was the chosen algorithm to calculate the communities, as it has been shown to have good summarization power and excellent execution time [54]; • **Edge Weight:** Given a number x, we remove all edges (u, v) with a weight less than or equal to x, $weight(u, v) <= x$. Removing "weak links," edges with a small weight allow a visual analysis of vertices with stronger connections; and • **Vertex Degree:** Given a number x, we remove all vertices v with degree less than or equal to x. Removing edges related to low degree vertices will allow the visualization to reveal vertices that interact with more vertices.

Figures 7 and 8 show six graphs created from the exact same vertices and edges, but using different reduction techniques. The data to create the graphs were extracted from the month of May, for retweet connections from the *Austin* dataset (More details about this dataset are available in Chapter VII). The original graph without any reductions has a total of 10,448 vertices and 11,977 edges. If no reduction is used, the graph is incomprehensible and does not produce useful insights, as shown in the first image of Figure 7. The *Vertex Degree* reduction technique with a large number will put the emphasis on the highest degree vertices, as shown in the third graph of Figure 8. In the *Community Percentage* reduction technique the graph emphasises vertices that are the centers of communities. To better understand the impact of the reduction techniques to the original graph, we have conducted extensive analysis, as presented in Chapter VII. The measures calculated are (i) the number of vertices, (ii) the number of edges, (iii) the average degree vertex, (iv) density, (v) the power nodes score, and (vi) the similarity

between the top 1% of vertices between the reduced and original graph. Figure 45 in Chapter VII shows the comparison of the graph reduction experiments.

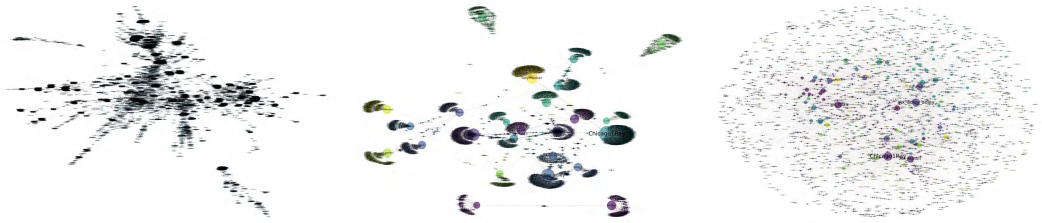

Figure 7: Sample graph reductions using the *Vertex Degree* and the *Community Percentage* techniques. *Austin* dataset: the original graph with no reduction (left); the *Vertex Degree* graph outcome: filtered all edges connecting two vertices with degree less than 115 (center); the *Community Percentage* and the *Vertex Degree* graph outcome: every community found in the graph reduced by 70%, and then every edge with degree less than 25 removed (right). (High resolution in Figures A.1, A.3, and A.4).

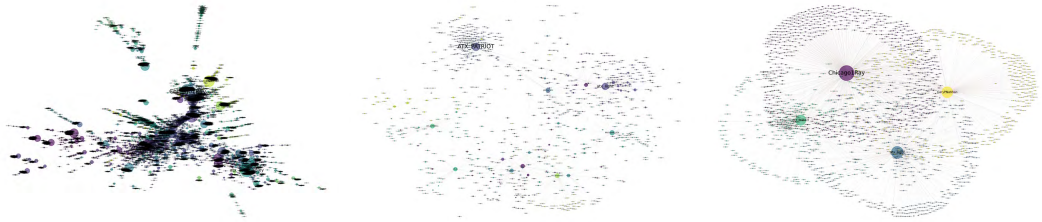

Figure 8: Sample graph reductions using the *Edge Weight*, the *Vertex Degree* and the *Community Percentage* techniques. *Austin* dataset: the original graph with no reduction with the size of the vertices scaled by degree (left); the *Edge Weight* and the *Community Percentage* aggregation: every edge with weight 1 removed, followed by the community reduction by 2% (center); the *Vertex Degree* aggregation: removing every edge connecting two vertices with degree less than 355 (right). (High resolution in Figures A.2, A.5, and A.6).

Discussions

Different ways of creating networks from Twitter data were proposed. Studying the different network types separately can be helpful for discovering different insights from the same data. The retweet connections network, for example, could potentially help highlight users with high influence [22]. Intuitively, we could also

reason that the replies network could help highlight users that can generate a lot of discussions, and the mentions network highlights people that are often brought into conversations, even if they were not necessarily participating in the discussions. The hashtag network can be helpful for finding communities of people discussing similar topics. The use of the hashtag network proved to be helpful for topic analysis, as shown in the details of Chapter IV and on the topic discovery experiments in Chapter VII. Scientists that are interested in studying Twitter data can choose what type of network fits better for their research question(s).

The clustering algorithms evaluated help us find communities within the networks and can be used for different objectives. The *Spectral Clustering* algorithm is helpful for cases where we are interested in how strong the relationship between vertices are. But *Spectral Clustering* has its limitations, as it has a high computational time and can only be applied to fully connected components. The *Louvain* method, on the other hand, which has a better computational time, was created specifically for the task of finding communities in the networks, and can be used for networks with separate connected components, but it does not consider the strength of the vertices connectivity and focuses only on maximizing the modularity of the network. Both the *Spectral Clustering* and the *Louvain* methods find disjoint communities. The novel super user centric approach, the *TN-Neighborhoods* method, finds overlapping communities around a set of high degree vertices, the super users. The *TN-Neighborhoods* method can be helpful for understanding the connections and discussions around particular influential vertices [57]. The different clustering performance evaluation metrics are helpful for understanding the characteristics of the networks, as they can show us how connected, dense, or well separated the communities in the network are. More experiments and discussions about the different performance metrics for each of the clustering algorithms are available in Chapter VII. The proposed reduction techniques allow the simplification of complex

and large graphs, making their visualization possible. The simplification comes with a cost, as some information gets lost with the reduction, so each technique can be useful for answering different questions. The *Community Percentage* technique should be used when we want to preserve the community structures and keep the emphasis on the centers of each community. The *Vertex Degree* technique should be used to keep the emphasis on vertices with the highest degrees. And the *Edge Weight* technique should be used to keep edges with the strongest connections. All the different techniques proposed can be used by themselves or in combination, so that the analysis can be shaped based on the available dataset, and towards answering a particular research question.

IV. TOPIC MODELING

Topic modeling (analysis and detection are often used as synonyms in this context) is a text mining technique that uses machine learning to automatically process text and categorize it by topic or subject. Topic modeling in Natural Language Processing (NLP) automates the identification of recurrent themes (topics) in the collection of text documents. Originally developed as a text-mining tool, topic models have been used to detect instructive structures in social networks. The optimized Latent Dirichlet Allocation (LDA) module is a popular generative statistical model for topic detection [24] that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. LDA relies on word frequency to find the mixture of topics for a given set of documents, and it was tuned to perform well on the natural language processing data of a set of large text documents.

Topic modeling in the age of social distancing has increased in relevance, as topic trending on Twitter arose as one of the most relevant resources of (mis)information. Increasing numbers of people rely on Twitter to receive news and form opinions on ongoing social, political, and economic policies, movements, and changes. Twitter data is different than a set of traditional documents. Tweets are a form of expressions that are short (up to 280 characters) and tend to be very colloquial in nature: acronyms, slang, misspellings, and bad or no punctuation are common. Thus, most of the research in LDA topic detection in Twitter datasets focuses on the use of different types of aggregations for input [26], or graph-based methods for clustering topics [27] to improve the topic discovery performance [25].

Hashtag analysis in Twitter data related to political consumerism [17] uncovers interesting trends. There are generally three types of political consumerism sentiment (i.e., political-, civic-, and consumption-related) that change the

sentiment of a tweet and its over arching purpose or relevance in the discussion [17]. The spread of digital wildfires in the form of misinformation, purposely fake and harmful tweets and quotes, have become a major concern in the past few years. Topic analysis helps separate misinformation tweets from the relevant ones, and researchers have recently published a dashboard of the daily list of identified misinformation tweets, along with topics, sentiments, and emerging trends in the COVID-19 Twitter discourse, as well as the spreading patterns of prominent misinformation tweets [64].

We propose to access the level of colloquialism in Twitter data, and to use tweet aggregation approaches outlined in Chapter III to improve discovery of Twitter dataset topics that are more human interpretable, and automate underlying analysis introduced in [17, 64] to help social scientists gain better insights into data patterns.

Pre-Processing Methods

A series of steps for cleaning the data and pre-processing the text were used before sending it as input to the topic model. Hashtags, user screen names, links, special characters, stop words, and numbers are removed. Then the words are lemmatized. Removal steps are as follows: (i) Remove links by cleaning everything that starts with *http*, (ii) remove hashtags by cleaning everything that starts with #, (iii) remove mentions by cleaning everything that starts with @,(iv) remove retweet symbols at the beginning of retweet messages, (v) remove any symbols, punctuation, return characters, extra spaces, special characters, and numbers, (vi) remove stop words, and (vii) lemmatize words.

Figure 9 shows a comparison of topics found before and after the cleaning steps using the same input.

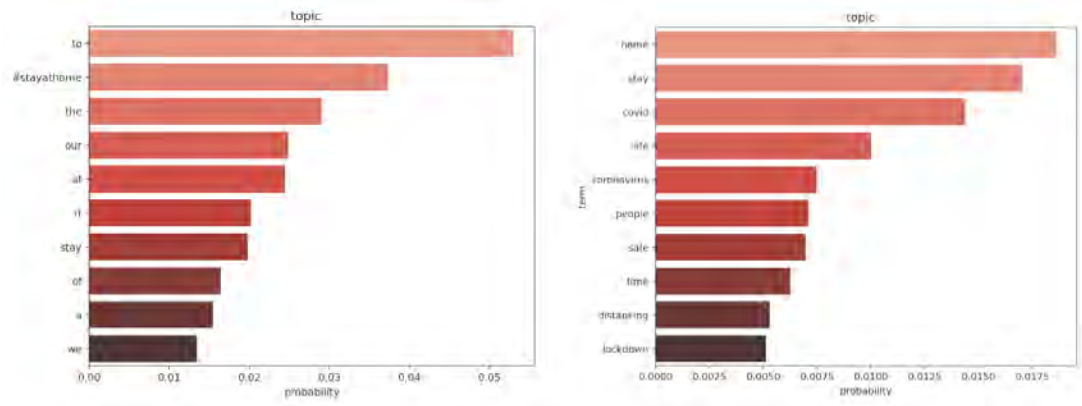

Figure 9: Sample topics discovered using the LDA model before (left) and after (right) using cleaning techniques. Data extracted from a community in the *Covid* dataset.

Topic Model

The *gensim* package [65] implementation was used for the topic model task, in particular the Optimized Latent Dirichlet Allocation (LDA) module [24].

Given a set of documents, the LDA model returns a set of topics, each with a set of words with a probability score that shows how likely each word can describe the topic. Such a model can be useful for predicting the topics of a new input text that was not part of the training set, classifying documents into categories, or retrieving insights about the main subject of a document. For the purpose of the work in this thesis, we are only interested in understanding what is being discussed with a given dataset, so all efforts in improving performance are towards that goal.

The following steps were done to train the models: (i) create a set of documents where each document corresponds to one tweet, (ii) pre-process the content of each document as explained in section *Pre-Processing Methods*, (iii) create a term dictionary of the corpus, where every unique term is assigned an index, (iv) convert the list of documents into a *Document Term Matrix* using the dictionary prepared in the previous step, (v) choose the number of topics, (vi) and train the model using the *gensim* package.

Evaluation

To evaluate the topics, human intuition and coherence metrics were used. Since the goal is to achieve human interpretability in the topics found, manual evaluation can be the best indicator, but we added the coherence metric c_v proposed in [66] to help in the evaluation of a large number of experiments. The c_v coherence was shown to outperform other existing coherence metrics [66].

Aggregation Methods

Large social networks can have a substantial amount of different parallel discussions. Even in the cases where the datasets were filtered based on a certain hashtag, different people can use the same hashtag for different discussions. The LDA model does an adequate job classifying topics for well defined documents, but it can be challenging to deal with the colloquial nature of Twitter data.

We propose an approach to pre-aggregate the tweets before using any topic prediction or visualization. The aggregation itself is already the first level of topic discovery, as it clusters the topics based on the specific aggregation.

The aggregation methods are as follows: (i) graph-based hashtag communities, (ii) graph-based user connection communities, (iii) and filters by time period, user's screen_names, or hashtags.

The graph-based *hashtag communities aggregation* uses the methods described in Chapter III to create a network of hashtag connections and separate them using the clustering techniques. Even though the hashtags themselves can be difficult to interpret, they can be great indicators of topics.

The graph-based *user connections aggregation* also uses the methods described in Chapter III to create a network of user connections and separate them using the clustering techniques. Users that are very connected to each other tend to share

similar topics.

The *filters* aggregation separates the tweets into groups that are bound based on the filter used. Time period filter, for example, separates the dataset into tweets that were created within a date range. That way it will be possible to see the topics being discussed for that specific period.

The three aggregation techniques can be used by themselves or in combination. The graph-based methods also have multiple options of clustering techniques and network creation, as explained in Chapter III. An example of this combination would be to aggregate the tweets based on the *Louvain* communities found in the replies only networks for a certain time period.

Visualizations

In order to visualize the topics for each of the approaches described in the previous sections, four different types of visualization were used: a LDA probability barchart, word clouds for the most frequent hashtags and words, barcharts showing the most frequent hashtags and words, and a hashtag connections graph.

Figures 10, 11, and 12 show a sample of a topic found for one of the communities in the *Austin* dataset. The text of the tweets were aggregated by the hashtag network and community *Louvain* method.

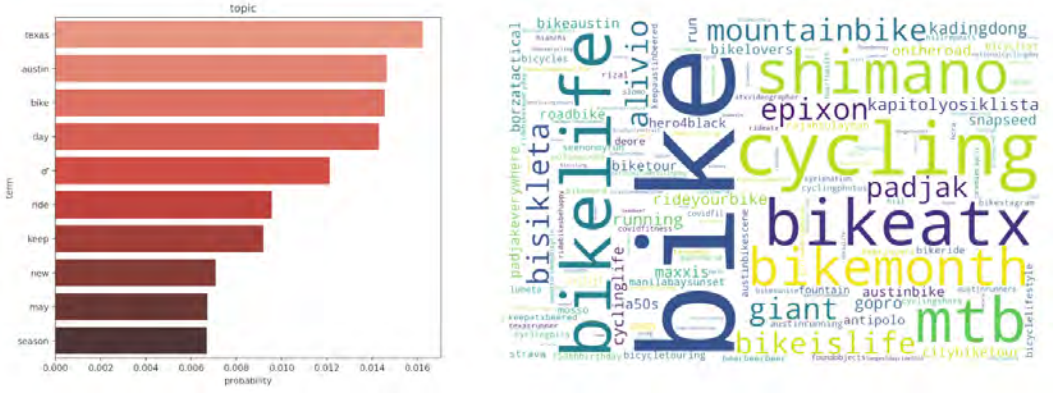

Figure 10: Visualizations for a topic discovered in the *Austin* dataset

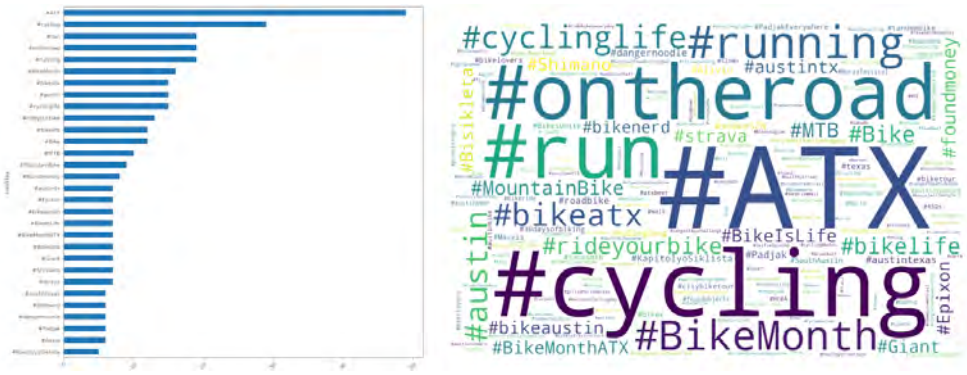

Figure 11: Barchart and word cloud visualizations for a topic discovered in the *Austin* dataset

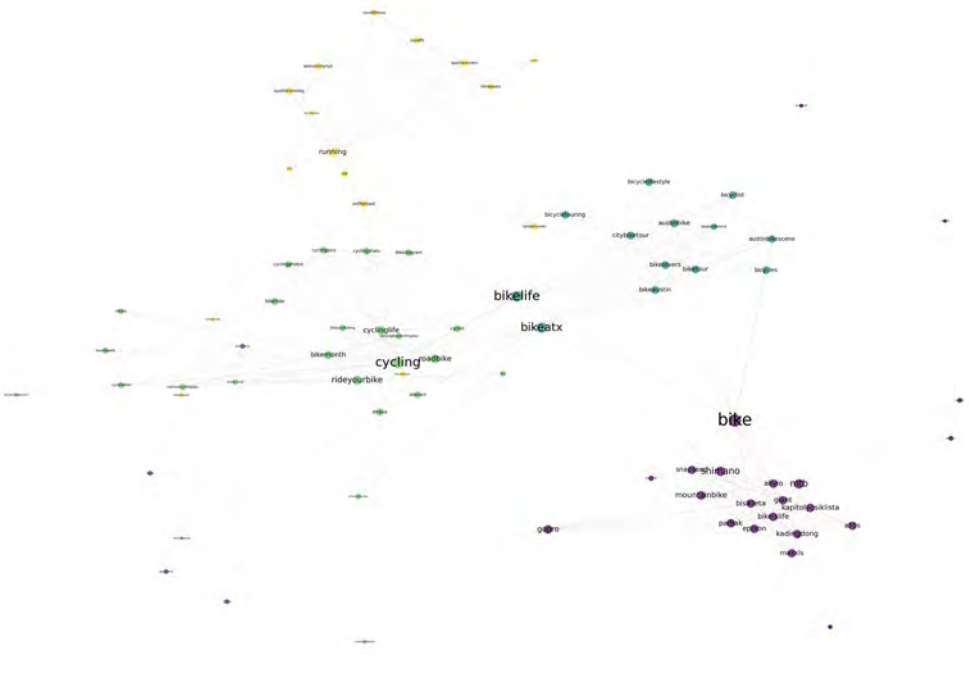

Figure 12: Sample hashtags graph visualization for a topic discovered in the *Austin* dataset

Colloquial Language Analysis

In order to understand the colloquialism of Twitter data and how it could affect the performance of traditional topic models, we investigate the readability ease of each tweet. There are different ways to score the readability of texts. The FK score (Flesch-Kincaid) is a common way of calculating this. Its formula takes into consideration the number of words, sentences, and syllables in a text. A slightly modified version of the FK score, as proposed on Davenport, 2014 [67], was used in order to accommodate Twitter's short format.

Flesch-Kincaid Score

The Flesch-Kincaid score is a readability test designed to indicate how difficult a passage in English is to understand. This score was used in this work to evaluate the readability ease of each of the tweets in the hope of finding and understanding the different levels of colloquialism of different tweets. Higher scores indicate greater ease in readability and lower scores indicate difficult or complex writing.

The standard FK score formula is:

$$206.835 - 1.015(\frac{\#Words}{\#Sentences}) - 84.6(\frac{\#Syllables}{\#Words})$$

The modified version proposed on Davenport, 2014 [67] uses the same concept, but it considers the entire tweet text as one sentence, and suggests a cleaning method to overcome the unconventional punctuation of the text.

Auxiliary Datasets

In order to get some additional information about the texts in the hope of getting more insight about how colloquial a text is, we also use the following datasets:

UrbanDictionary.com: A dataset with words from UrbanDictionary.com. Urban Dictionary is an online dictionary that captures crowd-sourced slang words and phrases. Even though it was originally focused on slang, now it can be used to define a more comprehensive set of words. This dataset was retrieved from Kaggle and includes 2.6 million words from UrbanDictionary.com.

Social Media Acronyms: A list of the most common social media acronyms was also used. A total of 140 acronyms were retrieved from an article on buffer.com [68].

Spelling Errors: A set of corpora were retrieved from an archive from the University of Birkbeck [69]. Three of their corpora were used: *Birkbeck* (36,133 misspellings of 6,136 words), *Aspell* (531 misspellings of 450 words), and *Wikipedia* (2,455 misspellings of 1,922 words).

NLTK Corpus - (Words): The NLTK [70] Words Corpus is the /usr/share/dict/words file from Unix, used by some spell checkers. It can be used to find unusual or misspelled words in a text corpus. This corpus was used to tag words as English or not.

FK Score Evaluation

The datset *MeToo* was used to evaluate the FK score for each tweet and to cross reference it with the additional datasets. More details about the *MeToo* dataset are available in Chapter VII. The tweets were split by the different ranges of FK scores, and percentages were calculated for each range based on the additional datasets. The results are presented in Table 4.

As shown in Table 4, the percentage of tweets with a least one acronym is higher for tweets with lower FK scores and the percentage decreases as the FK score increases. The percentage of total words with a least one acronym grows in the opposite direction, with greater percentages for higher FK scores. All FK score

Table 4: Tweets and word counts with % per FK score level. Data extracted from the *MeToo* dataset. FK score Level: The range the FK score; Tweets: the total number of tweets in millions; Words: the total number of words in millions; Acronym dic %T: The percentage of tweets with an acronym; Acronym dic %W: The percentage of words with an acronym; Not English %T: The percentage of tweets with a word that is not English; Not English %W: The percentage of words that are not English; Urban dic %T: The percentage of tweets with a word found in the Urban Dictionary; Urban dic %W: The percentage of words found in the Urban Dictionary; Spelling error %T: The percentage of tweets with a word found in the spelling error dataset; Spelling error %W: The percentage of words found in the spelling error dataset.

FK score	Total		Acronym dic		Not English		Urban dic		Spelling error	
level	Tweets	Words	%T	%W	%T	%W	%T	%W	%T	%W
<0	310	20,051	11.73	0.36	99.99	29.71	98.7	80.35	77.26	7.09
0-30	969	44,942	9.92	0.43	99.99	27.66	99.96	82.86	77.04	7.04
30-50	1,101	45,878	9.73	0.48	9.99	26.1	99.99	86.17	71.93	7.17
50-70	551	18,197	9.07	0.56	99.99	26.79	99.99	87.74	68.82	7.82
70-90	116	2,634	7.8	0.69	99.88	27.5	99.98	89.78	55.42	7.63
>90	37,310	505,833	5.89	0.88	99.63	31	99.95	91.71	37.26	7.16
Total	3,087	132,210	9.76	0.46	99.98	27.32	99.85	84.47	72.47	7.21

levels show a percentage of almost 100 for tweets with at least one word that was not recognized as English, and similar percentages between the different levels for the number of words in that same category. Almost 100% of tweets that had at least one word found in the UrbanDictionary, and a high percentage of total words as well. The total number of words found in the UrbanDictionary is also high, and it seems to increase as the FK score increases. The number of words found in the spelling error dataset was about 7% for the different FK score levels.

The auxiliary datasets were less helpful than initially expected. The UrbanDictionary dataset was not a reliable way to find slang words. The same word was saved with many different definitions and there were also many words that were not slang, but were still part of the dataset. Notice in Table 4 that a very large percentage of words were found in the slang dictionary, which makes the likelihood of every word actually being slang very small. The UrbanDictionary dataset turned out not to be a good indicator of colloquialism. The spelling error dictionary was

also not completely useful. Not a lot of words were able to be corrected properly. Finding better datasets in the future may improve the analysis.

Manually looking at some sample tweets for the different levels of FK score and the different values in Table 4, the results are actually counterintuitive. Low scores are supposed to be college graduate level and difficult to read, and high scores are supposed to be very easy to read. These levels did not seem to agree with the actual results. Very low and very high scored tweets had almost non intelligible text. Low to medium scores had the most intelligible sentences. And the different measures didn't seem to show a significant pattern between the different FK score levels.

Due to the nature of the tweets, it seems like the FK score does not translate on the same scale. But a unique scale could potentially be created for tweets if one wants to use this score to filter out non-useful text.

Even though this analysis didn't prove useful for a true understanding of the levels of colloquialism in Twitter data, it reiterated how challenging processing tweet texts can be.

Discussions

Different aggregation techniques and visualizations were proposed to improve the human interpretability of the topic discovery task. The different techniques can be used according to the research question. The *filters* technique can be useful for finding topics for a particular time range, a particular user, or hashtag. The graph-based *user connection communities* technique can be useful for finding topics that belong to a certain group of users that are related to each other. The graph-based *hashtag communities* aggregation technique was found to be especially helpful for separating the topics in a dataset.

The visualizations were also helpful in evaluating the topics being discovered. The graph visualization created from the connections between hashtags for a certain

topic proved to be particularly helpful in increasing human interpretability of the topics.

The topic coherence also showed improvements after the usage of the aggregation techniques. Details of the experiments done to evaluate the topics' coherence are available in Chapter VII.

Even though the proposed methods did make improvements in finding coherent topics, Twitter data still presents many challenges because of its colloquial nature. More efforts can be put into addressing those challenges.

V. DATA MANAGEMENT

Tweet data format is semi-structured. There might be a tag, mention, hashtag, text, reply, quote, image, link... or not. All the knowledge that tweets and the network of tweets contain cannot be easily analyzed and extracted using analytics tools designed for structured data. Traditional SQL data warehouses have been extended to support semi-structured data, and researchers have successfully used it as a supporting database for Twitter analysis [32, 33, 34, 35]. The problem with a relational database implementation for Twitter data is that it has a rigid schema definition making it difficult to adapt to the dynamic nature of Twitter data and the volume and scope of data analysis.

We propose to implement a NoSQL solution using MongoDB. MongoDB is a schemaless document-oriented database that has been proven to scale better for big-data, and perform better for inserts, updates, and simple queries [71]. The data does not have to have a particular structure limit, which increases the flexibility of the implementation for the dynamic fields from Twitter documents. Each document in MongoDB is capable of holding complex structures. MongoDB has been used before in different Twitter related research [72, 73, 74], and the overall processing pipeline has been explored in limited scope [75]. The challenge with a MongoDB implementation is that such databases do not perform well on aggregation queries, and do not have simple functionality for joining collections together [71, 76], as SQL databases do. We could use both systems for different parts of the pipeline, to take advantage of both SQL and NoSQL approaches. Implementation complexity does not warrant us to pursue this idea any further, as the system we have developed *scales* with millions of tweets, it *adapts* to the dynamic nature of Twitter data, it is *user friendly*, and *supports* data analytics. We propose to use MongoDB and take advantage of the flexibility with unstructured data, and to create multiple data

aggregation collections to address aggregated data retrieval and joins. The proposed approach scales and facilitates analysis while improving data retrieval performance.

Twitter Data Structure

A tweet document is comprised of multiple objects. The main object is called *Tweet* and it contains: (i) *static* information: data fields available for all tweets, even if empty, such as the tweet content, unique ID, post timestamp, and geo location if provided, and (ii) *dynamic* information: different fields of information that are available for different tweets and versions of client and server software used; there could be over 150 attributes associated with it [77]. The *Tweet* object is also a parent to other child objects, such as the *User* object that describes who authored the tweet and contains fields such as the username, user ID, the number of followers, and the account bio (even if empty); the *Place* object that exists when the tweet is geo-tagged; the *Entities* object that encapsulates arrays of hashtags, user mentions, URLs, and native media; and the *extended_entities* object for cases where there are attached media such as photo or video.

Objects retrieved at different dates and with different APIs can have a different set of fields; e.g. deprecated fields still exist in old retrieved files. The introduction of the *extended_tweet* field in 2017 holds 280 characters, an upgrade from 140. To make that transition work with backward compatibility, Twitter introduced the *extended_tweet* field that would contain the full message while the original tweet field would have the truncated information. So if the *extended_tweet* field exists, we should ignore the truncated values and use the extended ones. Two tweet documents can potentially have very different fields available, and Twitter's data dictionary explains each of the existing fields on the root-level *Tweet* object as well as the fields on the child objects [77]. Figure 13 illustrates an example of two tweet documents and their fields.

Figure 13: Two sample Twitter documents

Extracting Useful Information

Focused Tweet Info: In this study, we focus on the subset of fields from the original document, and treat them as an object. Core fields are always kept, and optional fields can be chosen based on the research question asked and the research need.

Tweet Message Field: The message of the tweet can be found (a) in the root of the tweet document, (b) in the *quote_status* or in the (c) *retweet_status* child objects in case the tweet was a retweet or contained a quote. The fields can be named *text*, *full_text*, or *extended_tweet.full_text*. We combine all available texts into a *text_combined* field, as having all content concentrated into one field facilitates later analysis on the tweet dataset topics.

Edges - Tweet Connections: Let x, y, z, w, and v be Twitter users, and let x be the user that created the tweet of interest. Edges are created between x and the other users when user x retweeted, quoted, replied to, or mentioned the other users. The following fields are used from the tweet document to create edges: the user object from the *Retweeted_Status* field, the user object from the *Quoted_Status*

field, the *user_ mentions* field from the entities object, and the *in_ reply_ to_ user_ id_ str* field. Figure 14 illustrates how the connections can be extracted from the tweet document, as our distinct connections can be established: *{x,y}* if x retweeted y, *{x,z}* if x quoted z, *{x,w}* if x mentioned w, and *{x,v}* if x replied to v.

Edges - Tweet Hashtags: Useful information can also be extracted from the connections between hashtags. Here, we extract pairs of hashtags that were used in the same tweet. So, for example, if a tweet contains the hashtags $\#x$, $\#y$, and $\#z$, we create the pairs *{#x, #y}*, *{#x, #z}*, and *{#y, #z}* to be used later in the pipeline for analyzing the graph of hashtag connections.

Words: The combined version of the message content is in *text_ combined* field, and we use NLTK [70] library to extract information about each word: if a word is in English or not, if it is a stop word or not, if it is a verb, noun, adjective, or adverb. The words are also broken down into syllables to help with the semantic analysis. Breaking the tweet messages into words provides a sentiment analysis base.

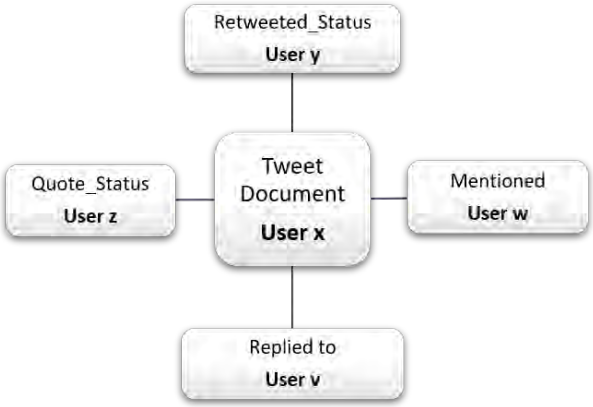

Figure 14: Various *Tweet* object fields are used to create different connections between users

Data Dictionary

The collection *tweet* stores the raw data from the Twitter JSON files, with no modification, and the collection *focusedTweet* stores the focused data extracted from the original documents. It is useful to have a separate collection for the focus data as it (i) decreases the amount of data stored in one collection and increases the query's performance, and (ii) enables core columns with a standard name processing. The field that contains the original tweet message can have different names, and standardizing that name in the focused collection streamlines the pipeline processing. We create a handful of collections to store cleaned and transformed data in the data pipeline process. Few of the collections share similar fields by design, as it enables post-processing queries to run easier and faster. Table 5 summarizes the core collections. Table 6 summarizes optional aggregate collections that store summary data for easy EDA. Table 7 summarizes the administrative collections. Administrative collections keep track of the records that have already been loaded and processed, save the searches that have already been done to the Twitter APIs, and control the recovery process. Table 8 shows the details of the temporary collections that get dropped and re-created to facilitate analysis and improve aggregate query performance. More details about each of the core collections are available in the Appendix section.

Table 5: Core collections

Core collections	
Collection	Description
tweet	This collection stores the full raw Twitter data. The fields available may vary, and are dependant on the content of the JSON files and APIs used as the source. A few additional fields were created in this collection to keep track of what file the tweet came from, the load timestamp, and to facilitate the recovery process.
focusedTweet	This collection stores only the most interesting information about the tweet. The definition of *interesting* can be different depending on the research question, so settings can be updated to drive what fields are interesting. Some core fields will always be available no matter the settings.
tweetWords	This collection stores each word separately for every tweet, and some additional information about the tweet. It also includes interesting tags about that word. (e.g English or not, verb or not, etc.)
tweetConnections	This collection stores the edges that connect two tweets together, either by retweets, quotes, replies, or mentions. It contains information about the tweet, where the connection came from, and about the two users that were connected. This collection is later used to build the edges for the graph analysis.
tweetHTConnections	This collection stores the edges that connect two hashtags together. If two hashtags were used in the same tweet, a record will be created in this collection. It contains information about the tweet where the connection happened, and about the two hashtags. This collection is later used to build the edges for the graph analysis.
users	This collection stores interesting information about the tweet user. Some core fields will always exist, but similar to the *focusedTweet* collection, settings are available to drive what fields are considered *interesting*. The same user can appear multiple times in a dataset with different values; for example, the same user can exist with two separate descriptions. This collection will not store multiple records for the same user and it will have only the first values found for each user. Other records for the same user will be ignored.

Table 6: Aggregate collections

Aggregate collections	
Collection	Description
agg_tweetCountByFile	This collection stores the count of all tweets in the dataset by files loaded.
agg_tweetCountByLanguage	This collection stores the count of all tweets in the dataset by language.
agg_tweetCountByMonth	This collection stores the count of all tweets in the dataset by month and year.

Table 7: Administrative collections

Administrative collections	
Collection	Description
adm_loadStatus	This collection stores the status of each collection's load. and how many tweets have already been inserted. In the case of a failure, it will be used to know which records have already been inserted and which ones haven't. This was created as part of the recovery process.
adm_loadedFiles	This collection stores the directory and file names that have already been loaded to the database. The load timestamp and the file path are the columns available. This is to make sure the same file doesn't get loaded multiple times. This was created as part of the recovery process.
searches	This collection stores all the requested searches to the Twitter APIs. The fields in this collection will vary, and will depend on the the result returned from the API. The time of the search and the name of the API used will always be available.

Table 8: Temp collections

Temp collections	
Collection	Description
tmpEdges	Temporarily saves edges of a graph so they can be used as lookup values in different queries.
tmpEdgesTweetIds	Temporarily saves tweets' Ids that refer to the edges saved on *tmpEdges*. This collection is used as lookup values in different queries.
tmpEdgesHTFreq	Saves the hashtags used in tweets for certain edges (Ids saved on *tmpEdgesTweetIds*). This is used in aggregate queries to count the frequency of hashtags for specific edges.
tmpEdgesWordFreq	Saves the words used in tweets for certain edges (Ids saved on *tmpEdgesTweetIds*). This is used in aggregate queries to count the frequency of words for specific edges.
It would be possible to create queries without these temporary tables and get the same results, but when querying a high volume of data it becomes nearly impossible to return aggregate queries using lookups from multiple tables in a reasonable amount of time.	

Recovery Process

The pipeline includes a recovery logic to make sure that the processes don't have to be re-run from the beginning in case of a failure. While inserting into the *tweet* collection, a sequence number gets attached to each tweet. Then every time any processes need to run, that sequence number is used to control what has already been processed or not. If something fails, the logic will be able to identify the last sequence number processed and continue from there. That logic is available for all the core collections and is driven by the values stored on *adm_loadStatus*.

DB Performance Tuning

In order to deal with the performance challenges while inserting and retrieving data from the collections, a series of techniques were used to improve the execution time.

InsertOne vs InsertMany: MongoDB gives users the flexibility to insert records one by one (insertOne method), or insert multiple records at a time (insertMany). When you insert records one by one, a request to change the collection gets sent to the database and the completion status is returned back for every single record. That means if we are inserting 3 million records, the database will be hit 3 million times, which could take some time. It is possible to solve that problem by inserting all records at the same time, that way the database would get hit only once. The problem with that solution is that in order to save all records at once, all data must be stored in memory first and then saved in the database. To avoid both problems and take into consideration the possible differences in hardware specification, a numeric parameter is used to drive how many records will be inserted at a time. For powerful computers, a large number may be appropriate, but a smaller number may be used for computers that are less powerful.

Indexes: Retrieving data for visualization purposes or for inserting into other collections can take a long time depending on the amount of data and filters used. To improve query performance, indexes were created in strategic fields that are used frequently and in many processes. The times to execute queries and to create the derived collections had significant improvement after creating the indexes. All indexes are created automatically when the collections get created. Table 9 show the list of all the indexes created.

Table 9: List of indexes created in the MongoDB collections

Collection	Field(s)
tweet	seq_no
focusedTweet	seq_no
focusedTweet	id_str
tweetConnections	tweet_id_str
tweetConnections	edge_screen_name_undirected_key
tweetHTConnections	tweet_id_str
tweetHTConnections	ht_key
tweetWords	tweet_id_str
tweetWords	tweet_seq_n
users	user_id_str & screen_name

Temporary Tables: Some of the more advanced queries that require lookups and aggregations can take too much time to run for large datasets. To deal with this challenge, temporary tables are created as part of the query requests to facilitate the lookup process without having to create nested aggregation in MongoDB. For example, when looking for hashtags that were used by users from specific edges in a graph, we can first create a temporary collection with all the edges for that specific graph. Then another temporary collection will save all the tweet Ids for those edges. With that information, data can be aggregated to count the hashtags used for those tweets. Even though it is possible to create one single query that will return that same result, it can be nearly impossible to return the result in a reasonable amount of time when dealing with a high volume of data. After implementing these

methods, the average execution time decreased by approximated 60%. Table 32 in Chapter VII shows the execution time improvements after implementing each of these methods.

VI. DATA ANALYTICS PIPELINE

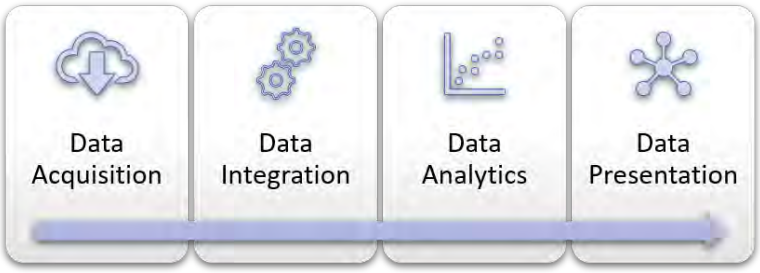

Figure 15: Data pipeline

In this section, we present the underlying design of the pipeline, and provide functionalities in *pytwanalysis* package [1]. The package provides a streamlined way for users to access the functionalities of all pipeline blocks. Twitter data analysis pipeline consists of several sequential analytics blocks, as illustrated in Figure 15. In the *Data Acquisition* block, tweets are downloaded from the source and saved as JSON objects. The criteria for acquiring data is pre-determined as described in Chapter VII, and motivated by the research questions asked. Next, in the *Data Integration* block, the Twitter JSON objects are cleaned, organized, and loaded into MongoDB database. The *Data Analytics* block supports exploratory data analysis, and offers graph analysis, clustering, topic discovery, time series, and community modeling functionalities. The *Data Presentation* block supports result validation, visualization, exploration, and output automation methods. The pipeline is subject agnostic: data acquisition and processing are determined by the end user based on the research question they try to answer. The *pytwanalysis* package can be installed via *pip* and imported as:

```
pip install pytwanalysis
import pytwanalysis as ta
```

For illustration purposes in this chapter, we create an exemplar object *myAnalysis* of type *ta*. The *myAnalysis* object can now call all *pytwanalysis* methods. Parameter *base_path* specifies where all the output files will be saved, and *db* parameted specifies the MongoDB connection to be used for processing the data:

```
# initialize object of type ta.TwitterAnalysis
myAnalysis = ta.TwitterAnalysis(BASE_PATH, db)
```

Data Acquisition

Figure 16: Data acquisition block

The *pytwanalysis* package supports two data acquistion modes: (i) collect data using Twitter's API and (ii) collect data from offline Twitter JSON objects. For (i), the *pytwanalysis* package supports calls to three different endpoints of the Twitter's Search API: the 7-Day search product from the Standard API [46], the 30-day search product from the Premium API [47], and the Full-archive search product from the Premium API [48]. The end user needs to provide the following items to *pytwanalysis* package: *Authentication* - the authentication keys that can be generated by getting a Twitter Developer account [78]; *Twitter API endpoint* - the twitter API product to use, in our case, either the Standard 7-Day search, Premium 30-day search, or Premium Full-archive search; and *Query* - the query that will be used to filter the tweets. Twitter provides an extensive documentation explaining how to build queries, use available parameters, and generate authentication keys. The documentation also explains the limitations of each API and their rate limits

[79]. For (ii), offline support, the *pytwanalysis* package provides functionality to load the data from files instead of straight from Twitter.

pytwanalysis Data Acquisition Methods

API Request Methods:

Sample Standard API 7-day search call:

```
# send request to 7-day search API
response = myAnalysis.search7dayapi(
    consumer_key = '[key]',
    consumer_secret = '[secret]',
    access_token = '[token]',
    access_token_secret = '[token_secret]',
    query = 'austintexas OR atx OR austintx OR atxlife',
    result_type = 'mixed',
    max_count = '100',
    lang = 'en'
)
```

Sample Premium API - (30-day or fullarchive) search call:

```
# send request to premium API
response, next_token = myAnalysis.searchPremiumAPI(
    twitter_bearer = '[bearer]',
    api_name = '30day',
    dev_environment = 'myDevEnv.json',
    query = '(coronavirus OR COVID19) lang:en',
    date_start = '202002150000',
    date_end = '202002160000',
    next_token = None,
    max_count = '100'
)
```

The definition of the parameters used in the *search7dayapi* and *searchPremiumAPI* methods are available in the Appendix section in Table C.1.

The process is as follows: The methods *search7dayapi* and *searchPremiumAPI* will send a request to the API based on the given query; the API will return the JSON documents matching the results, or an error message if any; *pytwanalysis*

inserts the JSON documents into MongoDB; a record with the metadata of that search gets created for future reference. The premium search API will also return a token value that can be used for a next request with the same query, that way it is possible to avoid getting duplicate records.

Scheduled Processing Script:

The *pytwanalysis* package supports unbiased sampling of Twitterverse by issuing Twitter API requests as schedules using a .bat file.

Sample call to create the .bat file:

```
# create python script and .bat file for scheduled processing
 myAnalysis.create_bat_file_apisearch(
     mongoDBServer='mongodb://localhost:27017',
     mongoDBName='myDBName',
     file_path='C:\\Data\\myScriptsFolder\\MyScriptName.py',
     python_path='C:\\Users\\Me\Anaconda3\envs\myEnv\python.exe',
     consumer_key = '[key]',
     consumer_secret = '[secret]',
     access_token = '[token]',
     access_token_secret = '[token_secret]',
     query = 'austintexas OR atx OR austintx OR atxlife',
     result_type = 'mixed',
     max_count = '100',
     lang = 'en')
```

The *create_bat_file_apisearch* method expects the same parameters as the *Standard Search - 7-Day API* method, plus four additional parameters: (i) the MongoDB server, (ii) the database name that will be used to insert the new tweets, (iii) the path where the script files should be saved, (iv) and the path where the *python.exe* is installed. Note that the package provides this extended functionality for the 7-day search API only. It will create two files; a .bat file and a python script containing the code necessary to make the requests. The end user determines the sampling schedule and may have to change the *.bat* file to include more steps in case additional logic is required for respective operating systems. Windows users can call the .bat file from the Windows *Task Scheduler*.

Loading JSON Files:

For the situation where there are physical JSON files saved in a certain path, the *pytwanalysis* package provides functionality for loading those files into MongoDB. The method expects the path where the JSON files are stored.

Sample call:

```
# load tweets from JSON files
myAnalysis.loadDocFromFile('C:\\Data')
```

The path of the files get stored in MongoDB as well so that it can be used to make sure the same file does not get loaded twice.

Data Integration

Figure 17: Data integration block

The *Data Integration* block is in charge of extracting useful information from the raw Twitter data, cleaning, organizing, and loading the data into MongoDB. The details of the structures created in MongoDB to support the pipeline are described in Chapter V.

Setting the Definition of Interesting

In order to tell the pipeline that certain fields in the Twitter data are useful for analysis, the package provides a method to configure what fields the end user

considers important. The settings will be used when loading the data into the *focusedTweet* collection. (Refer to Chapter V for more details).

Sample call to configure the fields that are interesting:

```
# set configuration to decide the fields to keep
focusedTweetFields='lang;retweet_count;in_reply_to_screen_name'
focusedTweetUserFields='name;description;location;friends_count;verified'
myAnalysis.setFocusedDataConfigs(focusedTweetFields, focusedTweetUserFields)
```

The following fields are always kept independent of the end user's choice: *id_str, text, quote_text, retweeted_text, hashtags, created_at, and user_id.*

And the following fields are kept as default but can be overwritten by setting the fields in the configuration: *lang, in_reply_to_status_id_str, in_reply_to_screen_name, user_screen_name, and user_name.*

Cleaning Methods

In order to improve the performance of the semantic analysis, the *ptwanalysis* package provides functionalities for cleaning the tweet messages based on the findings from the topic analysis portion of this thesis. The messages get cleaned while creating the collections in MongoDB to avoid having to execute the same steps multiple times. The cleaned messages get stored in a separated field. Hashtags, user screen names, links, and special characters are removed from the text field. Stop words are not removed, but get flagged as such and can be removed if necessary later.

Removal steps:

- Remove links by cleaning everything that starts with *http*.
- Remove hashtags by cleaning everything that starts with #.
- Remove mentions by cleaning everything that starts with @.
- Remove retweet symbols at the beginning of retweet messages.

- Remove any symbols, punctuation, return characters, extra spaces, and special characters.

For topic analysis it can also be useful to exclude numbers and stop words, but that piece only happens in the semantic analysis step.

Even though the cleaning happens as the collections get created, it is also possible to call the cleaning method manually.

Sample call:

```
myAnalysis.cleanTweetText('re The text to Clean for @Jane!!! :) #python')
```

The method receives a text as parameter and returns a clean version of that text. For the example above, the method would return:

```
'The text to Clean for'
```

The original information does not get lost, and mentions, hashstags, and links can still be retrieved if necessary.

Loading Data into MongoDB

With the raw information loaded into the database as part of the *acquisition block*, new collections are created to facilitate future data retrieval.

The *build_db_collections* method is in charge of extracting, cleaning, and loading the data into all the collections in MongoDB. The parameter *inc* is used to determine how many tweets will be processed at a time - the default is 100000. A large *inc* number may cause out of memory errors, and a low number may take a long time to run, so the decision of what number to use should be made based on the hardware specification.

Sample call:

```
# Load all data into all collections
inc = 10000
myAnalysis.build_db_collections(inc)
```

The different collections can also be built separately in case there is no need to build them all at once. The following are the separate methods that are executed as part of *build_ db_ collections*: *loadFocusedData*, *loadUsersData*, *loadWordsData*, *loadTweetConnections*, and *loadTweetHTConnections*. Refer to the package documentation [2] for more details on how to use each of these methods.

The recovery process explained in Chapter V is embedded into the methods that load the data into MongoDB. No separate method calls are needed in order to make sure the recovery process is activated.

Data Analytics

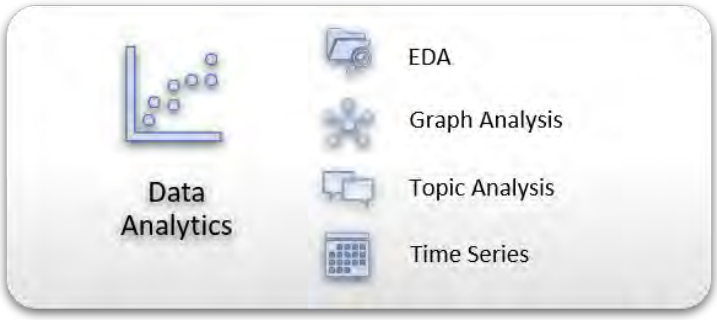

Figure 18: Data analytics block

With all the collections in MongoDB loaded with data, the next step is to start understanding the data, and to generate the different analysis components.

Exploratory Data Analysis

The pipeline has logic to print a summary of the initial exploratory data analysis for any dataset. The details of the metrics included in the output of method *eda_ analysis* are available in the Appendix section in Table C.2.

Sample call:

```
# print EDA & export EDA results to file
myAnalysis.eda_analysis()
```

Sample output:

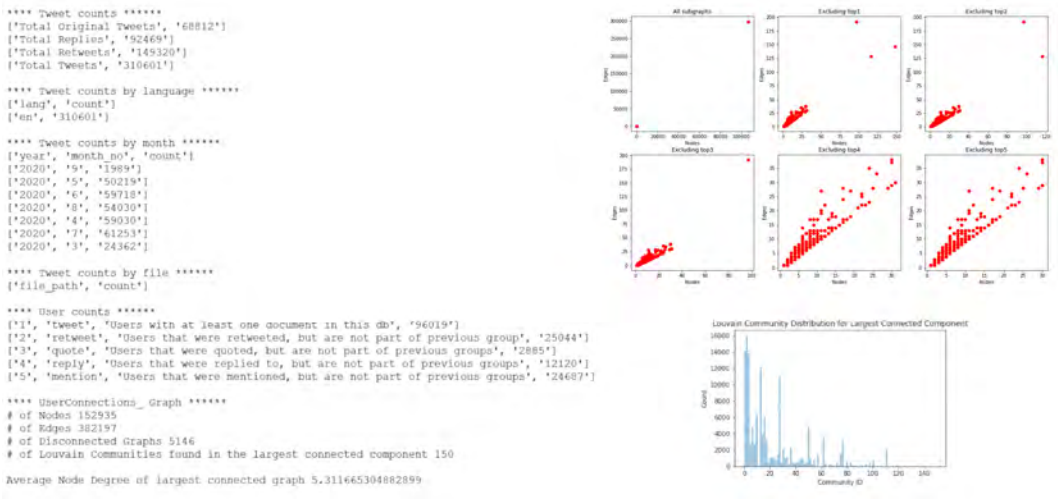

Figure 19: EDA sample output for the *Austin* dataset

Graph Analysis

The pipeline uses the steps for analyzing the datasets' social network and their communities by using the methods described in Chapter III. The pipeline can create the different networks, export measures for the graphs, export lists of vertices and edges for the different networks, compress and plot graphs, and print the different cluster metrics.

The first step is to create the edge files that will be used for the creation of the graphs. There are different options for generating the edges: *user_conn_all*, *user_conn_mention*, *user_conn_retweet*, *user_conn_reply*, *user_conn_quote*, and *ht_conn*. Refer to Chapter III for details about the difference between these types of graphs.

Sample call:

```
# in case there is a need to create edge files for separate periods
# Set period_arr=None in case there are no periods to analyze
period_arr = [['P1', '10/08/2019 00:00:00', '10/15/2019 00:00:00'],
              ['P2', '01/21/2020 00:00:00', '02/04/2020 00:00:00'],
```

```
                ['P3', '02/04/2020 00:00:00', '02/18/2020 00:00:00']]

# export text files with edge information
myAnalysis.export_mult_types_edges_for_input(
    period_arr=period_arr,
    type_of_graph='user_conn_all')
```

Given an edge file created from the Twitter data, a networkX graph G can be created.

Sample call:

```
# Load graph from edge file
G = myAnalysis.loadGraphFromFile(edge_file_path='[edge file path]')
```

Then, given a graph G, we can generate text files with the graph vertices and their respective degree, and the graph edges and their respective weight (method *nodes_edges_analysis_files*). We can cluster the vertices using the *Louvain* Community algorithm (method *calculate_louvain_clustering*). We can cluster the vertices using the *Spectral Clustering* algorithm (method *calculate_spectral_clustering*). We can remove the isolated vertices, exclude self-loops, and remove the disconnected graphs, keeping only the largest connected component (method *largest_component_no_self_loops*). We can remove edges of the graph based on the degree of the vertices connected to the edges (method *remove_edges*). We can remove a percentage of the vertices and the edges of each community of the graph (method *contract_nodes_commty_per*). We can create a sub-graph to represent the Neighborhood of vertex v (method *create_node_subgraph*). We can print some metrics about the graph, the number of vertices, number of edges, diameter, radius, extrema bounding, and center vertices (method *print_Measures*). We can print metrics about a community: separability, density, average clustering coefficient, clique size of the graph, number of maximal cliques in the graph, power nodes score, and average degree of all vertices (method *print_cluster_metrics*).

Refer to the package documentation [2] for more details and other available graph functionalities.

Topic Detection

The pipeline uses the steps for analyzing the topics within the messages and communities by using the methods described in the topic analysis portion of this thesis. The pipeline will run the LDA model and print the visualization with the results, it will also save text files, word clouds, and barcharts with hashtags and word frequency lists.

Given a text file we can create an array of documents. Every line becomes a document in the array.

Sample call:

```
# create one array of docs using a text file as source
myAnalysis.get_docs_from_file(file_path)
```

Given a document, in our case a tweet message, we want to add extra cleaning before training the LDA model. The method can remove numbers, remove stop-words, and lemmatize words in the documents. The end user can decide what level of cleaning they want to do.

Sample call:

```
# clean documents for topic analysis
myAnalysis.clean_docs(
    doc,
    delete_numbers=True,
    delete_stop_words=True,
    lemmatize_words=True)
```

Given a text file and the number of topics, we want to train a topic model that can predict the topics being discussed. The end user can decide the type of model that will be used for prediction: lda, lsi, or both. The end users can also decide if they want to save the trained models in a file for future use, and if they want to

save the results in MongoDB or in a text file. It also lets the end user decide the methods of cleaning they want to use in the document.

Sample call:

```
# train model from file. Model_type options: LDA, LSI, or both
myAnalysis.train_model_from_file(
    file_path='C:\\Data\\tweet_messages.txt',
    num_topics=4,
    model_name='MyModel',
    blnSaveinDB=False,
    blnSaveTrainedModelFiles=False,
    txtFileName=None,
    model_type='lda',
    lda_num_of_iterations=150,
    delete_stop_words=True,
    lemmatize_words=True,
    delete_numbers=True)
```

For the cases where the user already has an array of documents instead of a text file, they can alternatively use the method *train_model*, that will do the same as *train_model_from_file*, but will use an array of documents as the source.

Data Presentation

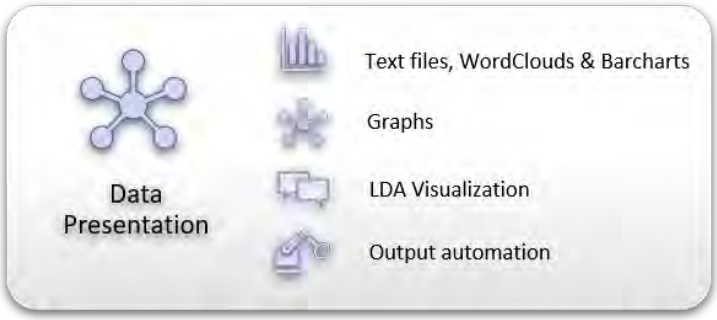

Figure 20: Data presentation block

The *Data Presentation* block supports data visualization options for the analysis done in the previous block. The end user can create word clouds, barcharts, different types of graphs, topic model visualization, and export data into text files.

Graph Visualization

Given a graph G, we can plot a representation of the graph using the *spring layout*. The end user can decide many of the details of the appearance of the graph, but they can also decide to just use the default values.

Sample call:

```
# visualize graph with details
myAnalysis.plotSpringLayoutGraph(
            G,
            v_graph_name='MyGraphName.png',
            v_scale=1,
            v_k=1,
            v_iterations=150,
            cluster_fl='N',
            v_labels=None,
            kmeans_k='',
            v_node_color='#A0CBE2',
            v_edge_color='#A79894',
            v_width=0.05,
            v_node_size=0.6,
            v_font_size=0.4,
            v_dpi=900,
            v_alpha=0.6,
            v_linewidths=0.6,
            scale_node_size_fl='Y',
            node_size_multiplier=6,
            font_size_multiplier=7,
            replace_existing_file=True)

# visualize graph using the default values
myAnalysis.plotSpringLayoutGraph(
            G,
            v_graph_name='MyGraphName.png',
            v_scale=1,
            v_k=1,
            v_iterations=150)
```

Table C.3 in the Appendix section shows the parameters expected in the

plotSpringLayoutGraph method. More details for these parameters are available in the *networkX* [80] library documentation.

Sample output:

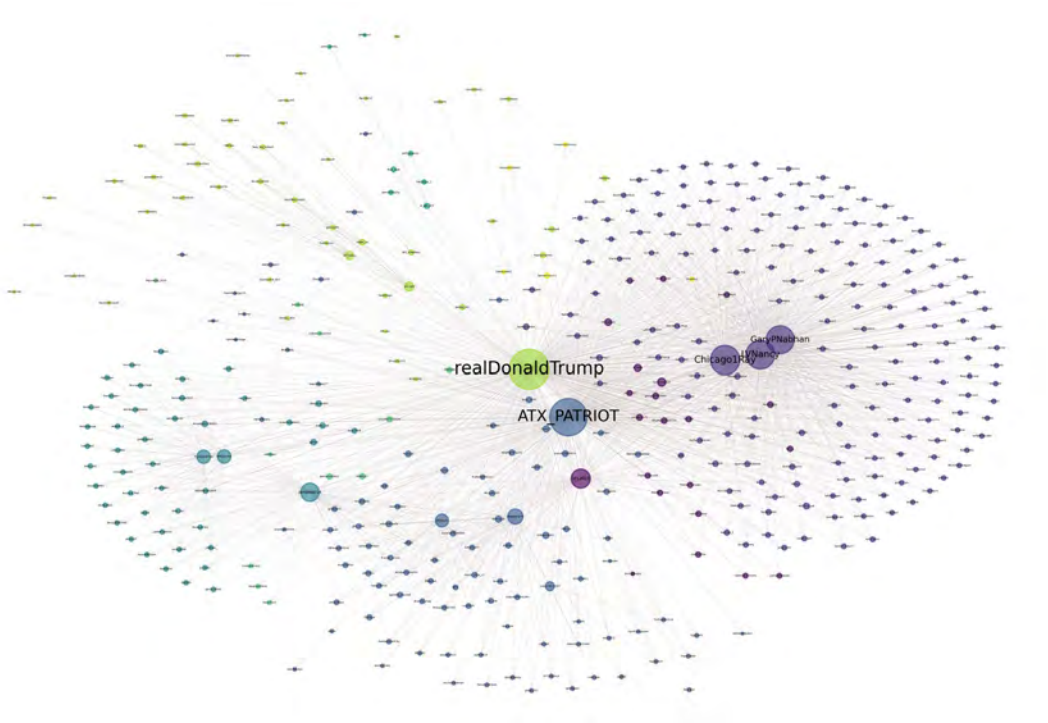

Figure 21: Sample graph output using the *Austin* dataset

Given a graph G, we can plot the distribution of the size of the different connected components in G. Each dot in the plot represents a separate component of the graph.

Sample call:

```
# plot the distribution of disconnected graphs
myAnalysis.plot_disconnected_graph_distr(
    G,
    file='C:\\Data\\ConnectedComponents-(Graphs).png',
    replace_existing_file=True,
    size_cutoff=None)
```

Sample output:

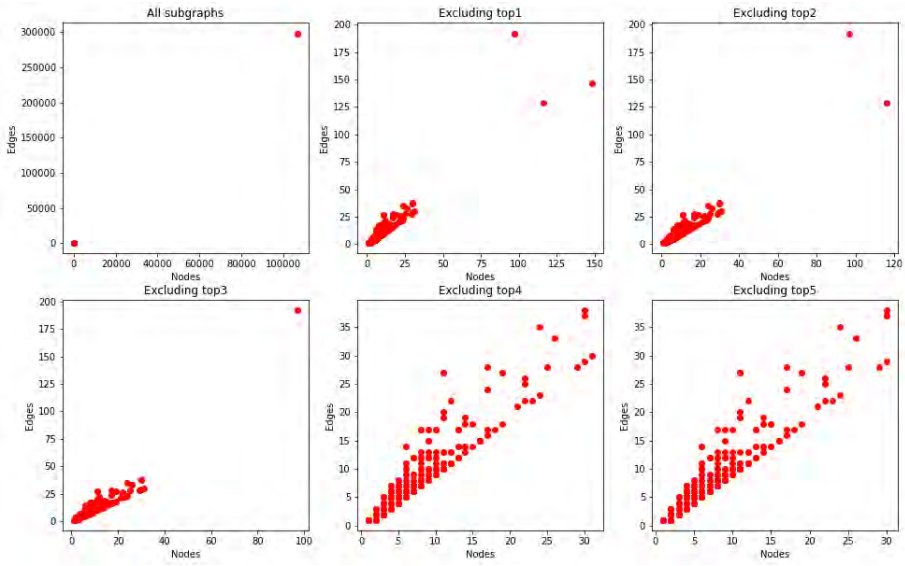

Figure 22: Sample distribution of the connected components of the graph extracted from the *Austin* dataset

Given a graph G, we can plot the distribution of the graph based on an attribute in G. After running the clustering algorithms, each vertex in G contains a label pointing to the community that they belong to. There are two available community related attributes: *community_louvain* and *spectral_clustering*. These attributes can be used to plot the distribution. This method can also be used to plot the distribution of manually added attributes.

Sample call:

```
# plot distribution of vertices based on graph attribute (e.g. community)
myAnalysis.plot_graph_att_distr(
    G,
    att='community_louvain',
    title='Community Counts',
    xlabel='Community ID',
    ylabel='Count',
    file_name=None,
    replace_existing_file=True)
```

Sample output:

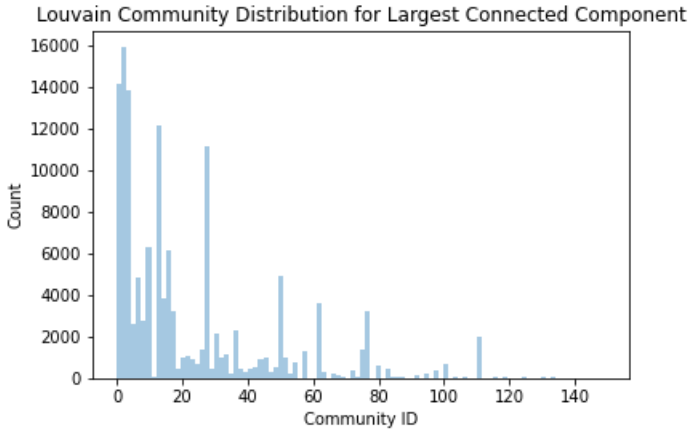

Figure 23: Sample *Louvain* community distribution for the *Austin* dataset

Word Clouds & Barcharts

Methods *ht_analysis_files* and *words_analysis_files* can be used to create barcharts and word clouds for words and hashtag frequency lists that are saved in the database. For the case where the frequency list is already saved in a text file, the methods *plot_word_cloud* and *plot_top_freq_list* can be called instead and the visualization will be based on the given frequency list.

Sample call:

```
# create word frequency list,
# barcharts, and word cloud files from data in the db
myAnalysis.words_analysis_files(
    path='C:\\Data',
    startDate_filter=None,
    endDate_filter=None,
    arr_edges=None,
    arr_ht_edges=None)

# creates hashtag frequency list,
# barcharts, and word cloud files from data in the db
myAnalysis.ht_analysis_files(
    path='C:\\Data',
    startDate_filter=None,
    endDate_filter=None,
```

```
    arr_edges=None,
    arr_ht_edges=None)
```

Refer to the package documentation [2] for more details and other available word clouds & barcharts functionalities.

Sample output:

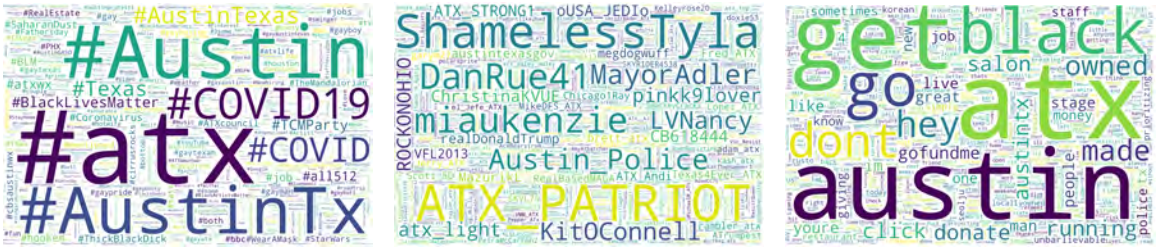

Figure 24: Sample word clouds extracted from *Austin* dataset

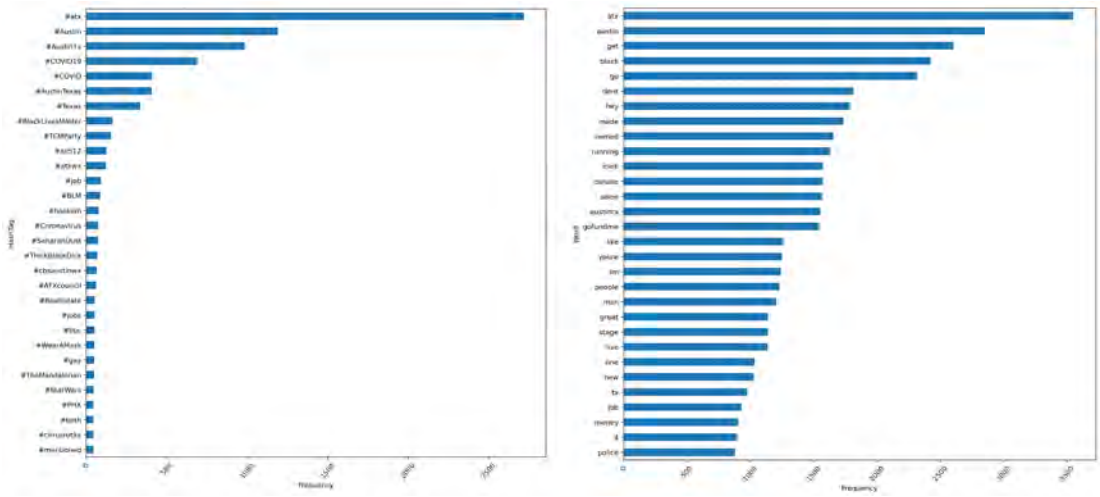

Figure 25: Sample barcharts for hashtags and word frequency extracted from the *Austin* dataset

Timeseries Graph

Every tweet document comes with a timestamp field that contains the time the message was created. That field is used in the pipeline to group tweets and topics into temporal buckets so that we can understand trends over a period of time.

The pipeline has logic that will plot a timeseries graph representing the tweet count by day, see example in Figure 26. It will also plot the count of how many times a hashtag has been used by day. See example in Figure 27 with the count for the top hashtags for the *Austin* dataset.

Sample call:

```
# create timeseries graph for tweet count and hashtag count
myAnalysis.time_series_files(
    path='C:\\Data',
    startDate_filter=None,
    endDate_filter=None,
    arr_edges=None,
    arr_ht_edges=None)
```

Sample output:

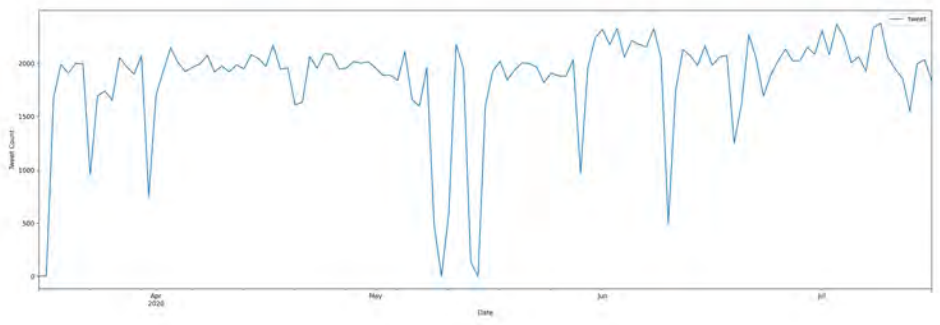

Figure 26: Tweet count by day for the *Austin* dataset

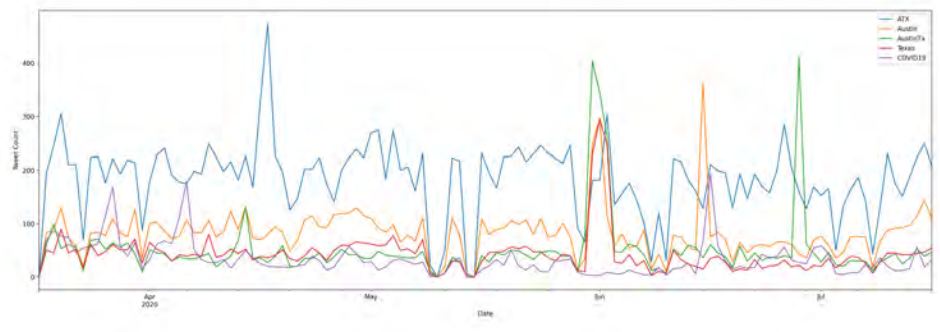

Figure 27: Top 5 hashtags count by day for the *Austin* dataset

LDA Model Visualization

After the LDA model has been trained, the pipeline can create a barchart showing the possible topics and their probability.

Sample call:

```
# plot graph with lda topics
myAnalysis.plot_topics(
    file_name='LDA_Vis.png',
    no_of_topics=4,
    model_type = 'lda',
    fig_size_x = 17,
    fig_size_y=15,
    replace_existing_file=True):
```

Sample output:

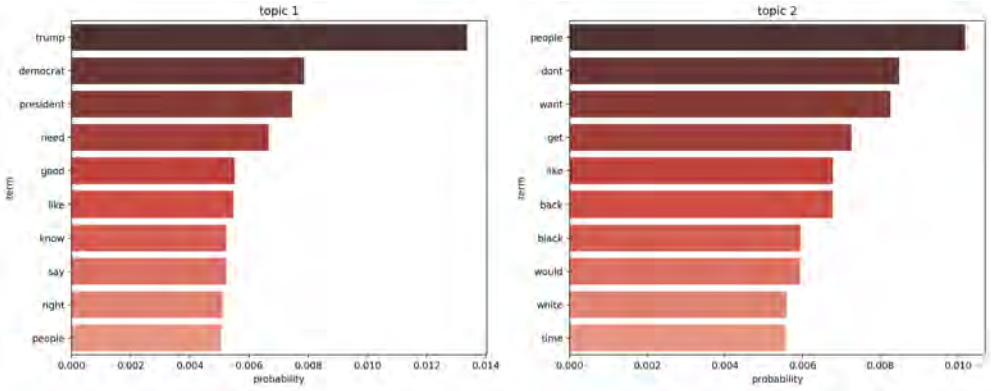

Figure 28: Sample output of the *plot_ topics* method

Exporting Data

Even though this pipeline will have different options of visualizing the graphs, one might want to use a different software for more visualization options. In that case, the pipeline will also have an option of exporting vertices, edges, user information, and other details to a tab delimited text file.

Sample calls:

```
# set the path where the files will be saved
exportPath = 'C:\\export_files\\'

# filter by period
start_date = '10/08/2017 00:00:00'
end_date = '10/09/2017 00:00:00'

# export clean tweet messages
myAnalysis.exportData(
        'text_for_topics',
        exportPath,
        startDate_filter = start_date,
        endDate_filter = end_date,
        replace_existing_file=False)

# hashtag frequency list
myAnalysis.exportData(
        'ht_frequency_list',
         exportPath,
         startDate_filter=start_date,
         endDate_filter=end_date,
         replace_existing_file=False)
```

Many other export functionalities are available. Refer to the package documentation [2] for more details.

Filters

Filters are available for each of the components of this pipeline. It is possible to filter by a specific period, by a hashtag, by an array of edges, or by the tweet being from a bot or not. For example, by using the edge array filter, every file generated will be for tweets that have the connections specified in the array. Frequency lists, graphs, lda model, etc. Filters can be combined or used individually.

Full Analysis Automation

Even though each small component of this pipeline can be executed separately, the pipeline also has an automation process to facilitate the complete analysis by creating all folders, edge files, and any other files based on given settings.

The Appendix section contains more details about the available configurations that determine which files will be generated in the automation, and with what characteristics (Tables C.4, C.5, C.6, and C.7). It also shows a list of possible files that will get created as part of the automation (Table C.8), and a sample of the folder structure that gets created after running the pipeline (Figure C.1).

Sample call:

```
# Set configurations
myAnalysis.setConfigs(
    type_of_graph=TYPE_OF_GRAPH,
    is_bot_Filter=IS_BOT_FILTER,
    period_arr=PERIOD_ARR,
    create_nodes_edges_files_flag=CREATE_NODES_EDGES_FILES_FLAG,
    create_graphs_files_flag=CREATE_GRAPHS_FILES_FLAG,
    create_topic_model_files_flag=CREATE_TOPIC_MODEL_FILES_FLAG,
    create_ht_frequency_files_flag=CREATE_HT_FREQUENCY_FILES_FLAG,
    create_words_frequency_files_flag=CREATE_WORDS_FREQUENCY_FILES_FLAG,
    create_timeseries_files_flag=CREATE_TIMESERIES_FILES_FLAG,
    create_top_nodes_files_flag=CREATE_TOP_NODES_FILES_FLAG,
    create_community_files_flag=CREATE_COMMUNITY_FILES_FLAG,
    create_ht_conn_files_flag=CREATE_HT_CONN_FILES_FLAG,
    num_of_topics=NUM_OF_TOPICS,
    top_no_word_filter=TOP_NO_WORD_FILTER,
    top_ht_to_ignore=TOP_HT_TO_IGNORE,
    graph_plot_cutoff_no_nodes=GRAPH_PLOT_CUTOFF_NO_NODES,
    graph_plot_cutoff_no_edges=GRAPH_PLOT_CUTOFF_NO_EDGES,
    create_graph_without_node_scale_flag=CREATE_GRAPH_WITHOUT_NODE_SCALE_FLAG,
    create_graph_with_node_scale_flag=CREATE_GRAPH_WITH_NODE_SCALE_FLAG,
    create_reduced_graph_flag=CREATE_REDUCED_GRAPH_FLAG,
    reduced_graph_comty_contract_per=REDUCED_GRAPH_COMTY_PER,
    reduced_graph_remove_edge_weight=REDUCED_GRAPH_REMOVE_EDGE_WEIGHT,
```

```
    reduced_graph_remove_edges=REDUCED_GRAPH_REMOVE_EDGES_UNTIL_CUTOFF_FLAG,
    top_degree_start=TOP_DEGREE_START,
    top_degree_end=TOP_DEGREE_END,
    period_top_degree_start=PERIOD_TOP_DEGREE_START,
    period_top_degree_end=PERIOD_TOP_DEGREE_END,
    commty_edge_size_cutoff=COMMTY_EDGE_SIZE_CUTOFF
    )
# create all files based on the given configurations
myAnalysis.edge_files_analysis(output_path=OUTPUT_PATH)
```

Scalability

Scalability was taken into account when choosing each of the techniques used in this pipeline. Performance tuning methods and the use of different parameters and filters was imperative for helping the analysis to focus on only what is considered interesting without wasting unnecessary time.

The subsection *Performance Tuning* in Chapter V shows some techniques used to improve the performance of the database insertion and data retrieval.

The creation of multiple filters and parameters also helps the end user choose only the pieces of information that are valuable to them. That way no time is wasted generating analysis for pieces of information that will not be used.

Another important method used for performance improvement purposes was exporting content into text files. While running the analysis files, some key data points get created and exported to text files. These files are then used later in the pipeline to load graphs, word clouds, and bar-charts. The time required to load data from text files is significantly lower than retrieving the data over and over again from the database.

The combination of these techniques allows the pipeline to be utilized even for larger datasets.

The Python Package

The *pytwanalysis* package is available for installation at:

`https://pypi.org/project/pytwanalysis/` [1]

The code is available in gitHub at:

`https://github.com/lianogueira/pytwanalysis` [81]

And the documentation of the *pytwanalysis* package is available at:

`https://github.com/lianogueira/pytwanalysis-documentation` [2]

The *pytwanalysis* package contains 4 classes:

- **TwitterDB**: This class will take care of all the MongoDB activities. It will load tweets from JSON files, create new collections, clean and transform data, query data, and export data.
- **TwitterGraphs**: This class will take care of graph related tasks. It will analyze graphs, export graph metrics, create sub-graphs, create clusters, calculate cluster metrics, plot graphs, and reduce graphs.
- **TwitterTopics**: This class will take care of topic discovery related tasks. It will train the LDA model, and print graphs with word and hashtag frequency.
- **TwitterAnalysis**: This class will inherit the methods of the other three classes and will be in charge of automating the creation of the analysis files and folder structure.

Tools

The following were the main tools used in this work:

Programming Language: Python 3.7

Database: MongoDB

Network Analysis: networkX library [52]

Topic Analysis: gensim [65] and NLTK libraries [70]

Word Clouds: wordcloud library

Visualizations: matplotlib library

Clustering: sklearn [56] & community libraries

VII. DATA AND EXPERIMENTS

Here, we apply the proposed pipeline to the analysis of the datasets with different sizes, formats, and languages. Goal is to access how the proposed software tool supports various Twitter datasets, fields available, and distribution of users and discussions. These datasets were used as case studies for the experiments. The datasets will be referenced as the following: **Austin**, a dataset with over 300,000 tweets that are related to the hashtags #austintexas, #atx, #austintx or #atxlife; **BJJ-en**, a dataset in English with over 200,000 tweets that are related to the hashtags #BJJ, #jiujitsu or #jiu-jitsu; **BJJ-pt**, a dataset in Portuguese with over 200,000 tweets that are related to the hashtags #BJJ, #jiujitsu or #jiu-jitsu; **Covid**, a dataset with over 8 million tweets that are related to the hashtags #Coronovavirus, #Covid19 or #Covid-19; **MeToo**, a dataset with over 3 million tweets that are related to the hashtag #MeToo; and **Random**, a dataset with over 100,000 random tweets that are not related to any particular subject.

Datasets

This section describes the six datasets used in the experiments, and the exploratory data analysis for each of them. Figures 29, 30, and 31 show a high level comparison between the counts in each of the datasets.

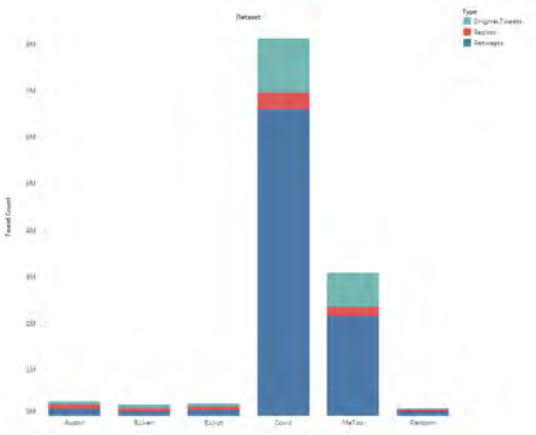

Figure 29: Tweet counts for all datasets

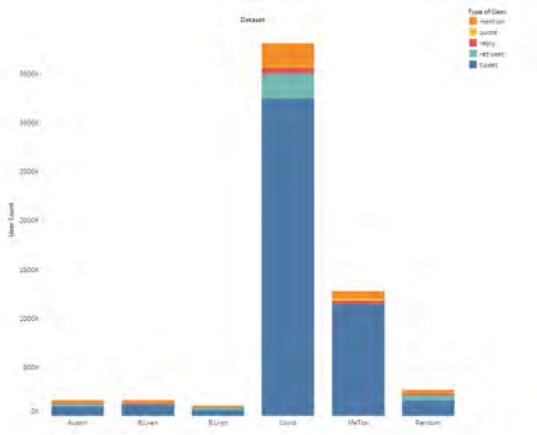

Figure 30: Unique users for all datasets

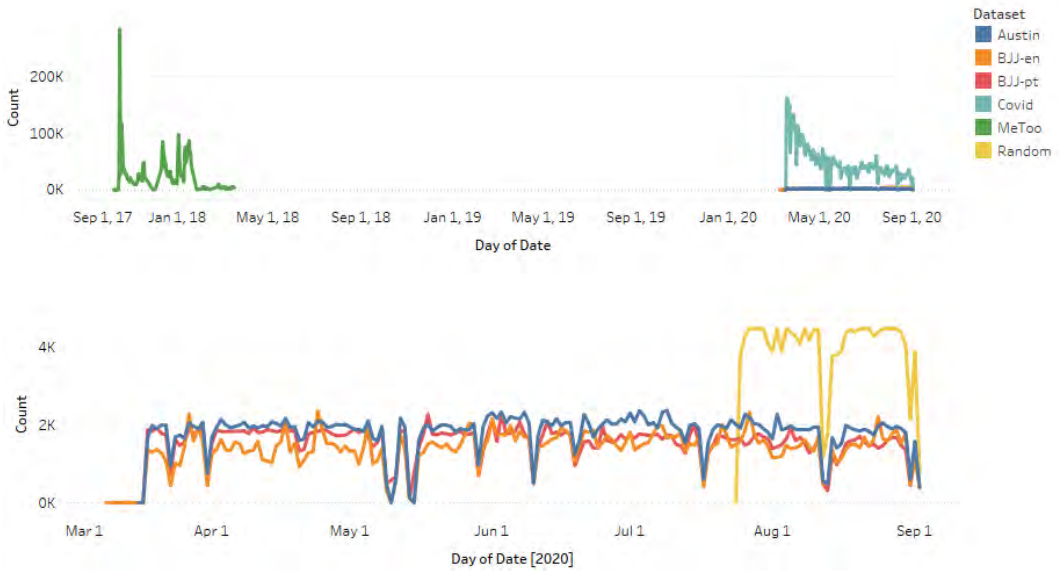

Figure 31: Tweet count by day for all datasets

Austin

This dataset contains a total of 310,601 tweets related to the hashtags #austintexas, #atx, #austintx, or #atxlife. All tweets in the dataset are in English and were created between March, 2020 and September, 2020. Within this data, there are almost 100 thousand users that either created original tweets, retweeted, quoted, or replied to other tweets.

Figure 32: Hashtag frequency - *Austin* dataset

Table 10: Tweet counts - *Austin* dataset

Tweet counts	
Original Tweets	68,812
Replies	92,469
Retweets	149,320
Total Tweets	**310,601**

Year	Month	# of tweets
2020	3	24,362
2020	4	59,030
2020	5	50,219
2020	6	59,718
2020	7	61,253
2020	8	54,030
2020	9	1,989
Total		**310,601**

Table 11: User counts - *Austin* dataset

Type of user		Count
tweet	Users with at least one document in this db.	96,019
retweet	Users that were retweeted, but are not part of previous group.	25,044
quote	Users that were quoted, but are not part of previous groups.	2,885
reply	Users that were replied to, but are not part of previous groups.	12,120
mention	Users that were mentioned, but are not part of previous groups.	24,687

BJJ-en

This dataset contains a total of 242,956 tweets related to the hashtags #BJJ, #jiujitsu or #jiu-jitsu. All tweets in the dataset are in English and were created between March, 2020 and September, 2020. Within this data, there are over 100

thousand users that either created original tweets, retweeted, quoted, or replied to other tweets.

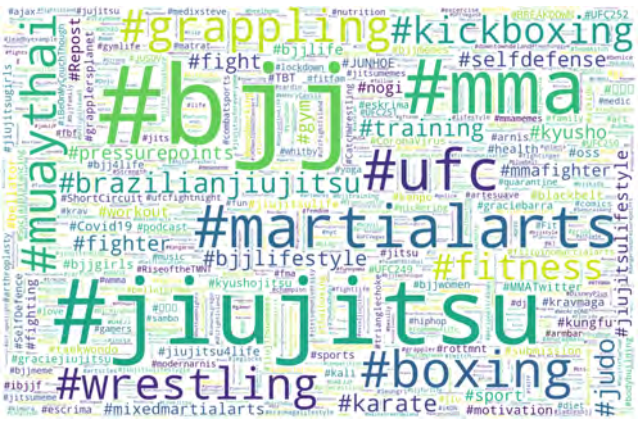

Figure 33: Hashtag frequency - *BJJ-en* dataset

Table 12: Tweet counts - *BJJ-en* dataset

Tweet counts	
Original Tweets	74,383
Replies	75,652
Retweets	92,921
Total Tweets	**242,956**

Year	Month	# of tweets
2020	3	18,066
2020	4	42,932
2020	5	38,361
2020	6	48,803
2020	7	49,828
2020	8	43,322
2020	9	1,644
Total		**242,956**

Table 13: User counts - *BJJ-en* dataset

Type of user		Count
tweet	Users with at least one document in this db.	108,845
retweet	Users that were retweeted, but are not part of previous group.	6,582
quote	Users that were quoted, but are not part of previous groups.	2,739
reply	Users that were replied to, but are not part of previous groups.	20,416
mention	Users that were mentioned, but are not part of previous groups.	21,950

BJJ-pt

This dataset contains a total of 267,101 tweets related to the hashtags #BJJ, #jiujitsu, or #jiu-jitsu. All tweets in the dataset are in Portuguese and were created between March, 2020 and September, 2020. Within this data, there are over 50 thousand users that either created original tweets, retweeted, quoted, or replied to other tweets. The different language will allow us to test the pipeline for tweets with other than English content.

Figure 34: Hashtag frequency - *BJJ-pt* dataset

Table 14: Tweet counts - *BJJ-pt* dataset

Tweet counts	
Original Tweets	73,497
Replies	69,099
Retweets	124,505
Total Tweets	**267,101**

Year	Month	# of tweets
2020	3	22,785
2020	4	53,658
2020	5	45,933
2020	6	49,891
2020	7	49,918
2020	8	43,261
2020	9	1,655
Total		**267,101**

Table 15: User counts - *BJJ-pt* dataset

Type of user		Count
tweet	Users with at least one document in this db.	55,866
retweet	Users that were retweeted, but are not part of previous groups.	31,237
quote	Users that were quoted, but are not part of previous groups.	4,236
reply	Users that were replied to, but are not part of previous groups.	9,221
mention	Users that were mentioned, but are not part of previous groups.	7,487

Covid

This dataset contains a total of 8,123,104 tweets related to the hashtags #Coronovavirus, #Covid19, or #Covid-19. All tweets in the dataset are in English and were created between March, 2020 and September, 2020. Within this data, there are over 3.2 million users that either created original tweets, retweeted, quoted, or replied to other tweets.

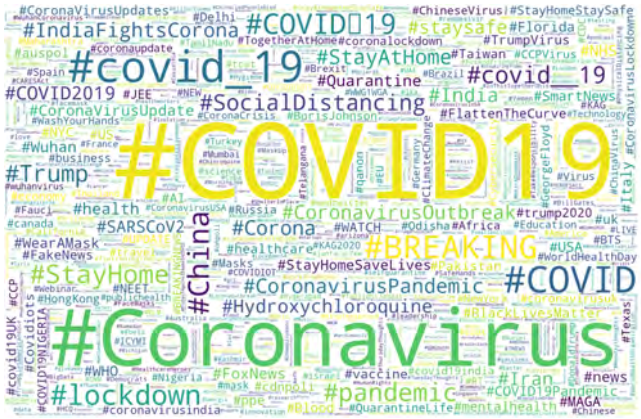

Figure 35: Hashtag frequency - *Covid* dataset

Table 16: Tweet counts - *Covid* dataset

Tweet counts	
Original Tweets	1,170,547
Replies	360,232
Retweets	6,592,325
Total Tweets	**8,123,104**

Year	Month	# of tweets
2020	3	1,651,431
2020	4	2,166,999
2020	5	1,261,279
2020	6	1,030,790
2020	7	1,159,296
2020	8	827,408
2020	9	25,901
Total		**8,123,104**

Table 17: User counts - *Covid* dataset

Type of user		Count
tweet	Users with at least one document in this db.	3,235,435
retweet	Users that were retweeted, but are not part of previous groups.	260,570
quote	Users that were quoted, but are not part of previous groups.	11,613
reply	Users that were replied to, but are not part of previous groups.	61,972
mention	Users that were mentioned, but are not part of previous groups.	237,199

MeToo

This dataset contains a total of 3,087,475 tweets that are related to the hashtag #MeToo. All tweets in the dataset are in English and were created between Oct, 2017 and March, 2018. Within this data, there are over 1.1 million users that either created original tweets, retweeted, quoted, or replied to other tweets.

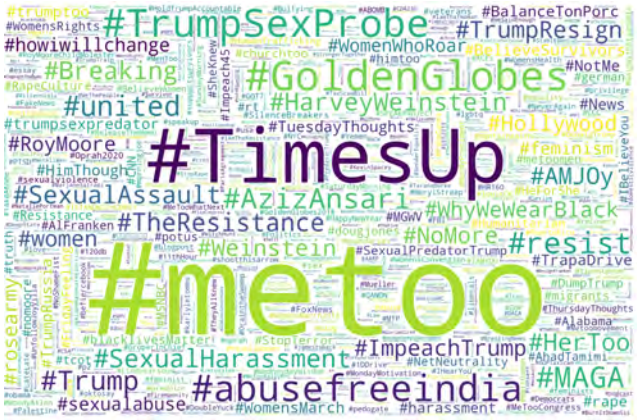

Figure 36: Hashtag frequency - *MeToo* dataset

Table 18: Tweet counts - *MeToo* dataset

Tweet counts	
Original Tweets	733,608
Replies	205,999
Retweets	2,147,868
Total Tweets	**3,087,475**

Year	Month	# of tweets
2017	10	870,775
2017	11	373,507
2017	12	723,944
2018	01	993,778
2018	02	75,553
2018	03	47,918
Total		**3,087,475**

Table 19: User counts - *MeToo* dataset

Type of user		Count
tweet	Users with at least one document in this db.	1,142,026
retweet	Users that were retweeted, but are not part of previous groups.	8,578
quote	Users that were quoted, but are not part of previous groups.	18,230
reply	Users that were replied to, but are not part of previous groups.	28,816
mention	Users that were mentioned, but are not part of previous groups.	78,044

Random

This dataset contains a total of 160,066 tweets that are not related to any particular subject. All tweets in the dataset are in English and were created between July, 2020 and September, 2020. Within this data, there are over 150 thousand users that either created original tweets, retweeted, quoted, or replied to other tweets. We

want to evaluate the performance of the methods proposed in this study for sparse datasets with a network that is not necessarily largely connected.

Figure 37: Hashtag frequency - *Random* dataset

Table 20: Tweet counts - *Random*

Tweet counts	
Original Tweets	24,548
Replies	31,984
Retweets	103,534
Total Tweets	**160,066**

Year	Month	# of tweets
2020	7	30,102
2020	8	125,287
2020	9	4,677
Total		**160,066**

Table 21: User counts - *Random* dataset

Type of user		Count
tweet	Users with at least one document in this db.	154,241
retweet	Users that were retweeted, but are not part of previous groups.	54,743
quote	Users that were quoted, but are not part of previous groups.	2,663
reply	Users that were replied to, but are not part of previous groups.	21,732
mention	Users that were mentioned, but are not part of previous groups.	35,042

Graphs

Different experiments were done to understand and compare the different types of network creation and the different clustering approaches. For more details on the methods and metrics used in the experiments, refer to Chapter III.

The terminology used in this chapter to represent the types of network is outlined in Table 22.

Table 22: Network creation terminology

Notation	Description
UC_A	All user connections.
UC_M	Mentions user connections only.
UC_RT	Retweets user connections only.
UC_RP	Replies user connections only.
UC_Q	Quotes user connections only.
HC	Hashtag connections.

Network Creation Experiments

In order to understand the different types of network creation, multiple networks were created using all six datasets with all the different network types.

Figures 38, 39, and 40 show the comparison between the different types of networks for each of the datasets. And Tables 23, 24, 25, 26, 27, and 28 show the details of the results. By observing the differences between the types of networks and datasets, we can draw some conclusions about the nature of the data.

As shown in Figure 38, the number of vertices and edges are larger for the *Covid* and *MeToo* datasets, which is expected since they have a larger number of tweets and users. The number of edges for each of the networks seems to follow a pattern, with the smallest network always being the *quotes* network, followed by the *replies* network, then *retweets*, *mentions*, and *all connections*. The only type of network that separates from the growth slope of the networks is the hashtag connections. The number of edges for hashtag connections grows more rapidly than all other networks. Another observation is that the slope of the growth for the *Random* dataset is slower than the others, which is expected for the *Random* dataset because the tweets were chosen randomly, so even as the number of vertices

(users, hashtags) increases, the number of edges between them won't increase as much since there are less connections between them.

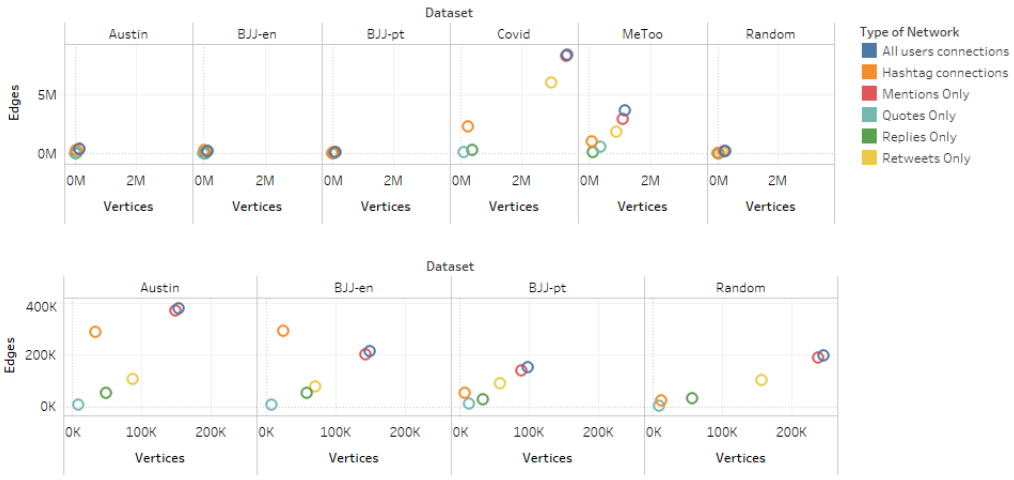

Figure 38: Number of vertices and edges for all datasets with the different ways of creating the networks. All datasets (top); Only smaller datasets (bottom).

Similarly, the average degree vertex for the different types of networks follows a similar pattern among the different datasets as shown in Figure 39, with the *hashtag connections* network having the highest average degree, followed by the *all connections* network, then *mentions*, *retweets*, *replies*, and *quotes*. As expected, the average degree vertex of the *Random* dataset is lower than all the others, since the tweets were chosen randomly, making it more unlikely that the users and subjects are connected. One interesting thing to notice is that even though the *Covid* and *Metoo* datasets are larger than the others, the average degree vertex is not that different from the other datasets.

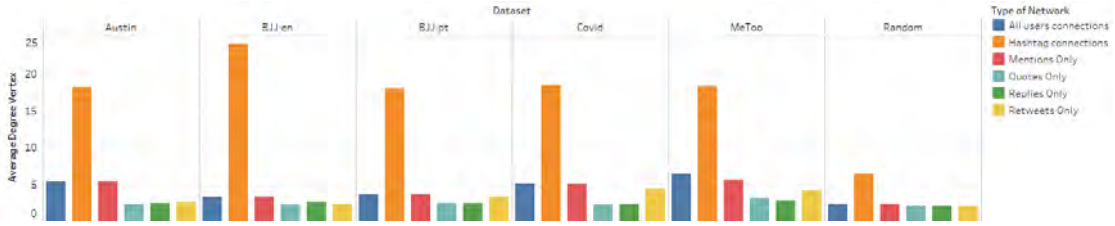

Figure 39: The average degree of the vertices for all datasets with the different ways of creating the networks

Another way of understanding how well connected a network is, is to find the number of disconnected sub-graphs within the main graph. Figure 40 shows how disconnected the networks are among the different datasets and network types. Sub-graphs that had less than 50 vertices were ignored in the analysis. The *Random* dataset has the largest number of disconnected components, which again proves that since the tweets were chosen randomly it is less likely that the users and subjects are connected.

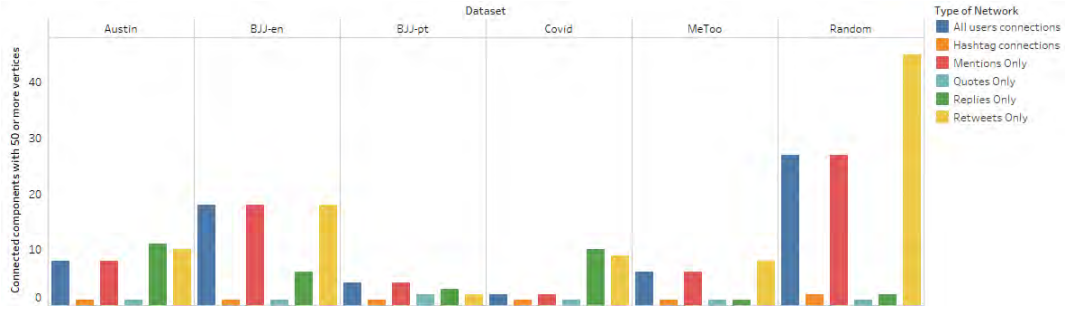

Figure 40: Number of separate connected components with at least 50 vertices for all datasets with the different ways of creating the networks

Table 23: *Austin* dataset graphs. Type: type of network; m: # of vertices; n: # of edges; LCCm: # of vertices in the largest connected component; LCCn: # of edges in the LCC; deg(T1): the degree of the most connected vertex; avgd: the average degree vertex; DCC: #of disconnected components(components with 50+ vertices); LVC:# of *Louvain* communities.

Type	m	n	LCCm	LCCn	deg(T1)	avgd	DCC	LVC
UC_A	152935	382197	138299	367299	9865	5.31	5146(8)	163
UC_M	148747	371854	134570	359310	9829	5.34	4733(8)	153
UC_RT	87156	104723	75166	95645	3214	2.54	3642(10)	138
UC_RP	49273	52695	34108	41431	3934	2.42	6311(11)	95
UC_Q	8459	8185	5153	6038	285	2.34	1425(1)	50
HC	33475	290486	31692	287822	18352	18.16	687(1)	96

Table 24: *BJJ-en* dataset graphs. Type: type of network; m: # of vertices; n: # of edges; LCCm: # of vertices in the largest connected component; LCCn: # of edges in the LCC; deg(T1): the degree of the most connected vertex; avgd: the average degree vertex; DCC: #of disconnected components(components with 50+ vertices); LVC:# of *Louvain* communities.

Type	m	n	LCCm	LCCn	deg(T1)	avgd	DCC	LVC
UC_A	149631	214338	116147	184985	2407	3.18	11613(18)	237
UC_M	143855	203109	111386	177315	2405	3.18	10536(18)	225
UC_RT	71848	77399	56501	65529	2384	2.31	4113(18)	129
UC_RP	58255	53595	24617	31054	1305	2.52	14801(6)	130
UC_Q	9136	7166	3498	3744	175	2.14	2459(1)	47
HC	25716	293831	24291	291618	16315	24.01	504(1)	145

Table 25: *BJJ-pt* dataset graphs. Type: type of network; m: # of vertices; n: # of edges; LCCm: # of vertices in the largest connected component; LCCn: # of edges in the LCC; deg(T1): the degree of the most connected vertex; avgd: the average degree vertex; DCC: #of disconnected components(components with 50+ vertices); LVC:# of *Louvain* communities.

Type	m	n	LCCm	LCCn	deg(T1)	avgd	DCC	LVC
UC_A	97582	151783	75961	135597	3027	3.57	8954(4)	123
UC_M	89126	139251	71365	126993	2805	3.55	6904(4)	113
UC_RT	57140	88940	52227	85592	2780	3.27	1754(2)	71
UC_RP	32608	28784	13924	16520	473	2.37	8699(3)	115
UC_Q	13560	12406	6806	8338	732	2.45	2816(2)	58
HC	6447	51393	5547	49893	3624	17.98	292(1)	49

Table 26: *Covid* dataset graphs. Type: type of network; m: # of vertices; n: # of edges; LCCm: # of vertices in the largest connected component; LCCn: # of edges in the LCC; deg(T1): the degree of the most connected vertex; avgd: the average degree vertex; DCC: #of disconnected components(components with 50+ vertices); LVC:# of *Louvain* communities.

Type	m	n	LCCm	LCCn	deg(T1)	avgd	DCC	LVC
UC_A	3502121	8307228	3208768	8053031	127238	5.01	130470(2)	5122
UC_M	3459276	8154948	3176137	7942162	125338	5.00	120750(2)	2981
UC_RT	2991359	5989652	2673207	5785400	37570	4.32	138009(9)	2687
UC_RP	385982	333603	156648	179977	14985	2.29	113433(10)	290
UC_Q	116924	91305	40134	44127	1710	2.19	34698(1)	145
HC	252420	2247271	242357	2228628	154393	18.39	4316(1)	953

Table 27: *MeToo* dataset graphs. Type: type of network; m: # of vertices; n: # of edges; LCCm: # of vertices in the largest connected component; LCCn: # of edges in the LCC; deg(T1): the degree of the most connected vertex; avgd: the average degree vertex; DCC: #of disconnected components(components with 50+ vertices); LVC:# of *Louvain* communities.

Type	m	n	LCCm	LCCn	deg(T1)	avgd	DCC	LVC
UC_A	1170036	3644240	1120115	3576906	87063	6.38	21545	780
UC_M	1099694	2912935	1046430	2862548	83970	5.47	21212	593
UC_RT	919341	1862352	876094	1830406	82783	4.17	18214	480
UC_RP	150996	171676	96957	132245	4903	2.72	29345	127
UC_Q	389200	595269	363981	572835	14503	3.14	11251	166
HC	110789	1020348	110737	1015550	1142026	18.34	26	339

Table 28: *Random* dataset graphs. Type: type of network; m: # of vertices; n: # of edges; LCCm: # of vertices in the largest connected component; LCCn: # of edges in the LCC; deg(T1): the degree of the most connected vertex; avgd: the average degree vertex; DCC: #of disconnected components(components with 50+ vertices); LVC:# of *Louvain* communities.

Type	m	n	LCCm	LCCn	deg(T1)	avgd	DCC	LVC
UC_A	246162	197479	65089	76148	1968	2.33	65419(27)	263
UC_M	237641	188540	62731	73378	1904	2.33	61948(27)	206
UC_RT	156196	103446	9084	9142	341	2.01	53667(45)	88
UC_RP	56954	31909	294	293	289	1.99	28018(2)	5
UC_Q	8608	4706	97	96	51	1.97	4198(1)	10
HC	11228	24681	5387	16987	169	6.30	1875(2)	61

Clustering Experiments

In order to understand the different clustering methods, multiple experiments were done using all six datasets.

Figure 41 and 42 show a comparison of the execution time of each clustering method. The *Spectral Clustering* method clearly has the highest execution time, followed by the *Louvain* method, and then *TN-Neighborhoods*. Both the *Louvain* and *TN-Neighborhoods* methods seem linear, but the *TN-Neighborhoods* has a much slower slope. The *Spectral Clustering* method ran in the *MeToo* dataset for over 24 hours and then returned a memory error. The execution time experiments were done on a Windows machine, with the Intel(R) Core(TM) i-9900K CPU @3.60GHz, and 64G of RAM.

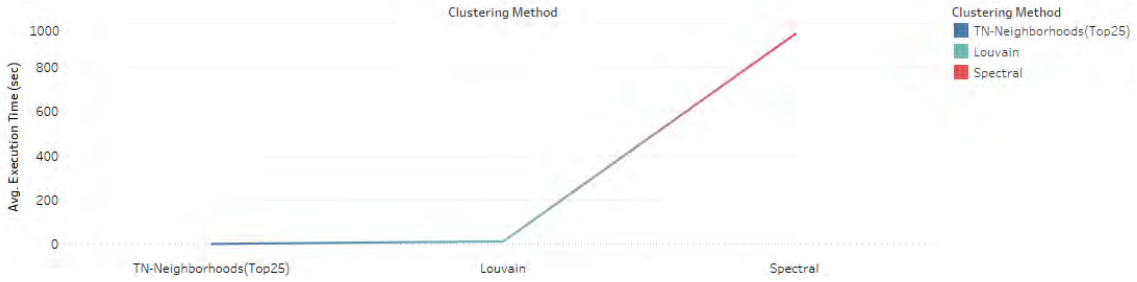

Figure 41: Execution time comparison for the different clustering methods

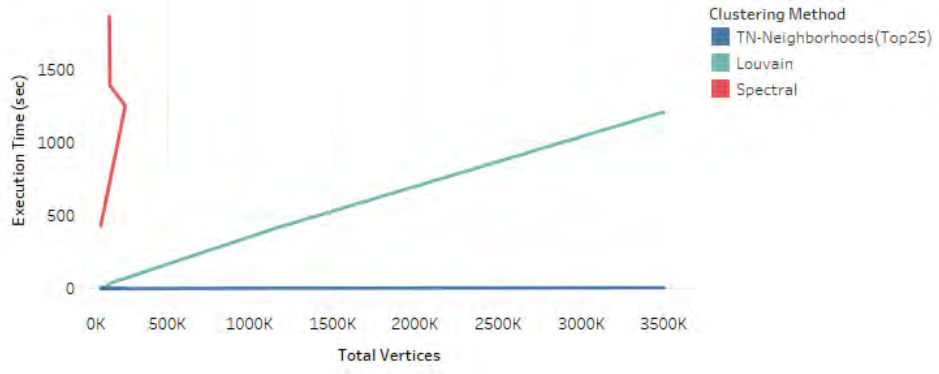

Figure 42: Execution time in seconds for each of the clustering methods as the number of vertices in the graph grows

Figure 43 shows the average metrics values for the three types of clustering methods, using all six datasets. The details of the results are shown in Table 29. By analysing the goodness metrics shown in Figure 43, it is possible to observe that the *Spectral Clustering* method outperforms the other methods in separability and density, followed by the *Louvain* method, and then *TN-Neighborhoods*. The average clustering coefficient is higher for the *TN-Neighborhoods* method, and performs similarly to the other two methods, with the *Louvain* method having slightly higher values. The *TN-Neighborhoods* method has the highest Power Nodes Score, followed by the *Spectral Clustering* method, and then *Louvain*. So, we can conclude that the *Spectral Clustering* method seems to have the most well separated and dense communities, the *TN-Neighborhoods* method the best well connected the neighborhoods, and the *Louvain* method sits in the middle of the two for all categories.

Figure 44 shows the metrics for the three types of clustering methods for the *Austin* dataset for all types of networks. The details of the results are shown in Table 30. The hashtag connections network has the highest score for separability and average clustering coefficient, but has lower density. Separability is similar for all other network types. All six types of networks follow similar values for the power nodes score.

Figure 43: Metric averages for the communities found using the different clustering methods. No results available for *Spectral Clustering* for the *Covid* and *MeToo* datasets due to the high execution time.

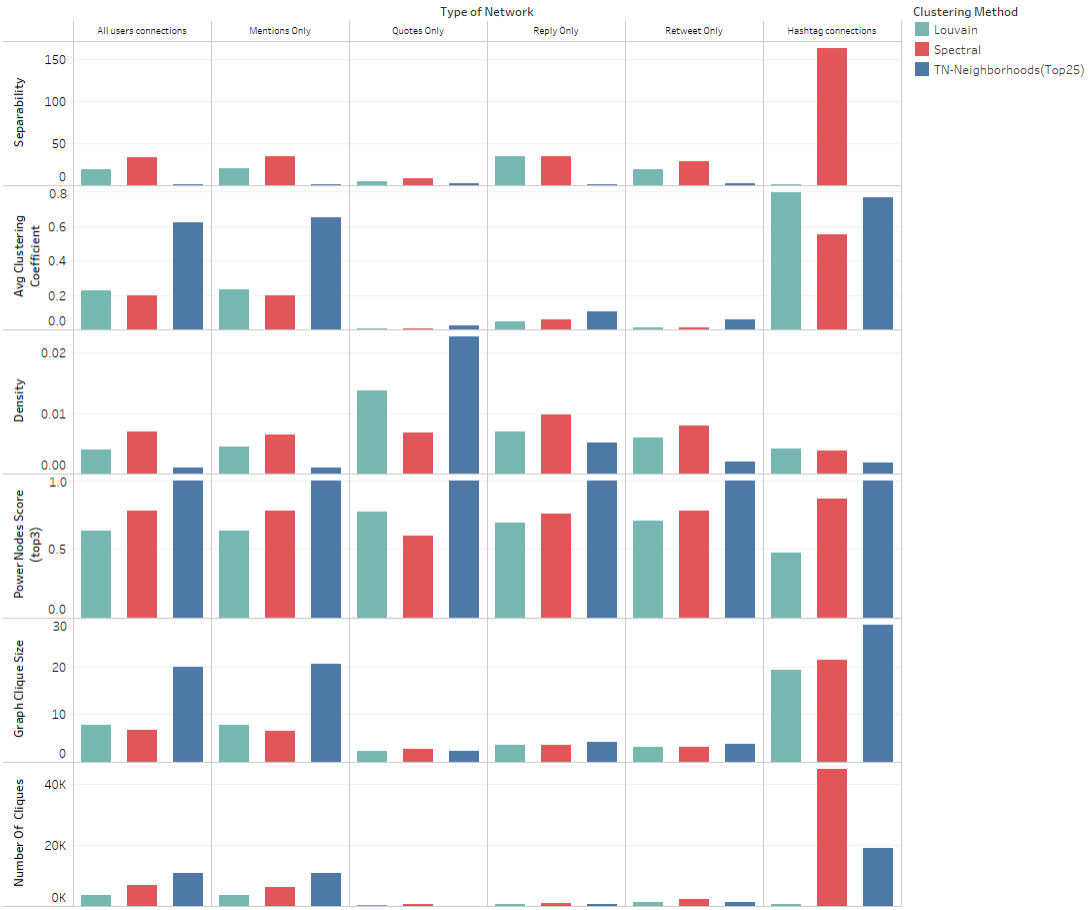

Figure 44: Metrics average of the communities found using the different clustering methods for *Austin* dataset for the different types of networks

Table 29: Different clustering methods measures. Dataset: the name of the dataset; C: the clustering method (S=*Spectral Clustering*, L=*Louvain*, T=*TN-Neighborhoods*); E(sec): the execution time in seconds; k(n>100): the number of clusters (the number of cluster with 100 or more vertices); ad: the average vertex degree; sep: separability; den: density; acc: average clustering coefficient; pns: power nodes score; gcs: graph clique size; nc: number of cliques.

Dataset	C	E(sec)	k(n>100)	ad	sep	den	acc	pns	gcs	nc
Austin	S	1388.4	148(36)	3.412	33.67	0.007	0.198	0.78	6.66	6919
Austin	L	17.9	148(66)	3.973	19.41	0.004	0.229	0.63	7.75	3436
Austin	T	0.3	25(25)	18.044	1.54	0.001	0.625	1.00	20.04	10907
BJJ-en	S	1864.2	238(63)	2.729	45.70	0.009	0.163	0.81	4.25	2238
BJJ-en	L	35.2	238(118)	2.750	32.85	0.005	0.156	0.58	4.97	1067
BJJ-en	T	0.2	25(25)	2.732	13.75	0.002	0.207	1.00	5.52	1106
BJJ-pt	S	427.9	121(17)	2.546	22.84	0.008	0.071	0.81	4.05	6486
BJJ-pt	L	7.1	121(70)	2.658	8.03	0.007	0.146	0.71	5.77	1173
BJJ-pt	T	0.2	25(25)	2.718	0.26	0.001	0.203	1.00	6.52	1724
Covid	S	Not available - too large								
Covid	L	1209.8	5517(96)	2.653	10.37	0.003	0.034	0.36	5.06	62793
Covid	T	6.7	25(25)	3.943	0.11	0.000	0.310	1.00	6.92	43321
Random	S	1250.9	265(55)	2.309	31.89	0.011	0.032	0.41	2.89	1170
Random	L	5.5	265(177)	2.172	20.69	0.008	0.017	0.36	2.58	375
Random	T	0.1	25(25)	2.122	0.78	0.009	0.062	1.00	2.84	324
MeToo	S	Not available - too large								
MeToo	L	418.7	806(68)	3.577	3.76	0.002	0.177	0.44	7.20	29399
MeToo	T	3.5	25(25)	6.418	0.21	0.000	0.612	1.00	10.44	52013

Table 30: Different clustering method measures for *Austin* dataset with the different types of networks. Type: the type of network; C: the clustering method (S=*Spectral Clustering*, L=*Louvain*, T=*TN-Neighborhoods*); E(sec): the execution time in seconds; k(n>100): the number of clusters (the number of cluster with 100 or more vertices); ad: the average vertex degree; sep: separability; den: density; acc: average clustering coefficient; pns: power nodes score; gcs: graph clique size; nc: number of cliques.

Type	C	E(sec)	k(n>100)	ad	sep	den	acc	pns	gcs	nc
UC_A	S	1388.4	148(36)	3.412	33.67	0.007	0.198	0.78	6.66	6919
UC_A	L	17.9	148(66)	3.973	19.41	0.004	0.229	0.63	7.75	3436
UC_A	T	0.3	25(25)	18.044	1.54	0.001	0.625	1.00	20.04	10907
UC_M	S	1271.1	157(38)	3.318	34.65	0.006	0.198	0.77	6.60	6352
UC_M	L	19.7	157(62)	4.181	19.75	0.005	0.232	0.63	7.83	3616
UC_M	T	0.3	25(25)	18.195	1.77	0.001	0.651	1.00	20.60	10873
UC_Q	S	4.8	50(5)	2.162	8.03	0.007	0.007	0.60	2.80	812
UC_Q	L	0.3	50(19)	2.053	4.21	0.014	0.005	0.77	2.31	173
UC_Q	T	0.0	25(25)	2.041	2.18	0.023	0.025	1.00	2.40	105
UC_RP	S	117.8	97(31)	2.229	35.04	0.010	0.058	0.75	3.54	1115
UC_RP	L	2.9	97(53)	2.198	34.09	0.007	0.049	0.69	3.56	638
UC_RP	T	0.1	25(25)	2.416	1.06	0.005	0.107	1.00	4.28	643
UC_RT	S	442.6	143(38)	2.171	28.46	0.008	0.014	0.78	3.15	2229
UC_RT	L	5.6	143(70)	2.149	19.07	0.006	0.013	0.71	3.21	1170
UC_RT	T	0.2	25(25)	2.233	2.05	0.002	0.058	1.00	3.76	1178
HC	S	1796.5	103(2)	11.091	163.42	0.004	0.550	0.86	21.50	44793
HC	L	11.4	103(27)	10.447	0.87	0.004	0.798	0.47	19.29	810
HC	T	0.2	25 (25)	31.042	0.43	0.002	0.768	1.00	28.84	18819

Graph Reduction Experiments

Here, we compare the graphs before and after the reduction. The number of vertices, number of edges, average degree vertex, density, power nodes score, vertices and edges compression percentage, and the top 1% vertices similarity calculations are compared. Figure 45 shows a comparison of the different reduction techniques and the effect caused on the resulting compressed graph. Table 31 details results of the experiments for a sample graph.

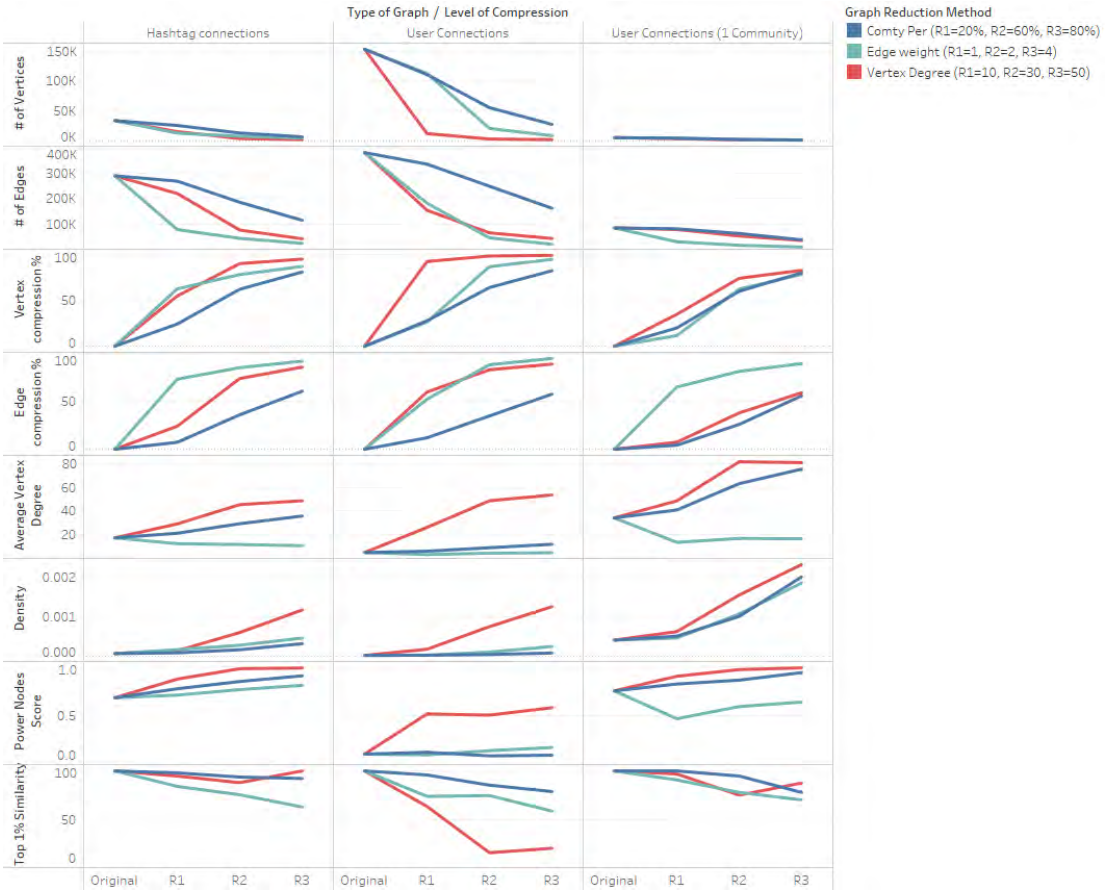

Figure 45: Comparison of the different reduction techniques for three graphs of different sizes. Data was extracted from the *Austin* dataset.

The number of vertices and edges decreases as the level of reduction for each method increases, as shown in Figure 45. The compression occurs more gradually for the *Community Percentage* method compared to the others. The average degree

Table 31: Graph reductions comparison. Original graph extracted from the users connections in the *Austin* dataset. GReduction(Par): Graph Reduction Method(ParameterUsed); n: number of vertices; m: number of edges; avgd: average vertex degree; den: density; pnc: power nodes score; ts: top1% similarity; n%: vertex compression percentage; m%: edge compression percentage.

GReduction(Par)	n	m	avgd	den	pnc	ts	n%	m%
Original	152935	382197	4.998	0.000013	0.106	-	-	-
Comty Perc(20%)	110561	336457	6.086	0.000018	0.125	95.83	27	11
Comty Perc(60%)	55153	249209	9.037	0.000036	0.088	85.29	63	34
Comty Perc(90%)	27210	162320	11.931	0.000074	0.095	78.67	82	57
Edge Rem Weight(1)	112375	182367	3.246	0.000018	0.098	73.73	26	52
Edge Rem Weight(2)	20403	45139	4.425	0.000098	0.141	74.51	86	88
Edge Rem Weight(4)	8292	19854	4.789	0.000241	0.174	58.53	94	94
Vertex deg Rem(10)	11774	154023	26.163	0.000170	0.515	63.24	92	59
Vertex deg Rem(30)	2674	65005	48.620	0.000748	0.503	15.38	98	82
Vertex deg Rem(50)	1592	42578	53.490	0.001257	0.579	20.00	98	88

vertex increases for the *Vertex Degree* method as the graph becomes more compressed. This behaviour is expected, since the vertices that are only connected to lower degree vertices are getting removed from the graph. The opposite seems to occur for the *Edge Weight* method. Since the edges with low weight are getting removed, vertices that had high connectivity because of the pendant vertices will have their degree lowered. The density of the graph actually increases as the reduction methods are applied, and the top vertices' similarity decreases. Figure 46 shows three graphs created from the exact same vertices and edges, but using different reduction techniques. The data to create the graphs were extracted from the month of May, for retweet connections from the *Austin* dataset. The original graph without any reductions has a total of 10,448 vertices and 11,977 edges. A larger version of the graphs in Figure 46 is available in the Appendix section.

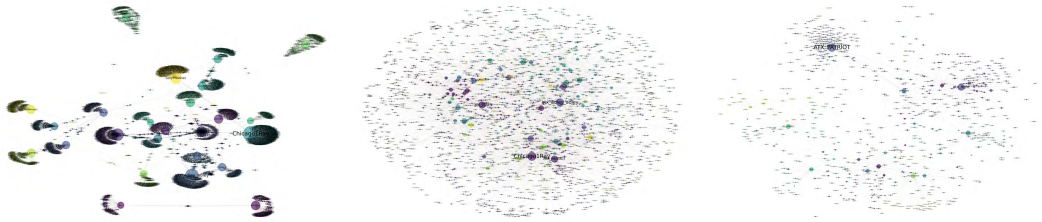

Figure 46: Graph reduction samples from the *Austin* dataset. *Vertex Degree* graph outcome: filtered all edges connecting two vertices with degree less than 115 (left); *Community Percentage* and the *Vertex Degree* graph outcome: every community found in the graph reduced by 70%, and then every edge with degree less than 25 removed (center). *Edge Weight* and the *Community Percentage* aggregation: every edge with weight 1 removed, followed by the community reduction by 2% (right); (High resolution in Figures A.3, A.5, and A.6).

Topic Discovery

Here, we access the quality of the topics found by experimenting with proposed types of aggregation methods (Chapter IV). First we vary the number of topics set for the LDA model. Figure 47 shows the c_v coherence metric change as the number of topics increases. The coherence appears to grow as the number of topics grows.

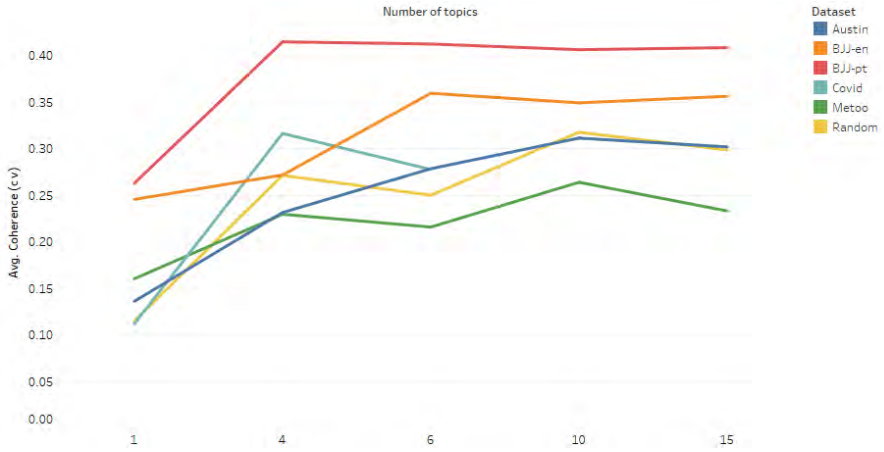

Figure 47: Coherence values as the number of topics grows. Experiments include multiple datasets.

Figure 48 shows the execution time for training the LDA model using different numbers of tweets for input. The training time seems to increase linearly as the

number of tweets increases. The execution time experiments were done in a Windows machine, with the Intel(R) Core(TM) i-9900K CPU @3.60GHz, and 64Gb of DDR4 RAM.

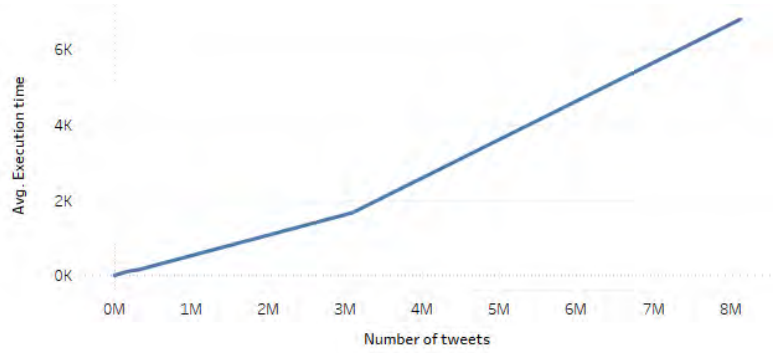

Figure 48: The execution time for training the LDA model using different numbers of tweets for input

Figure 49 shows the average coherence values for the different aggregation methods. The different datasets show slightly different behaviors for each aggregation method, but it's clear that the aggregation methods do increase topic coherence.

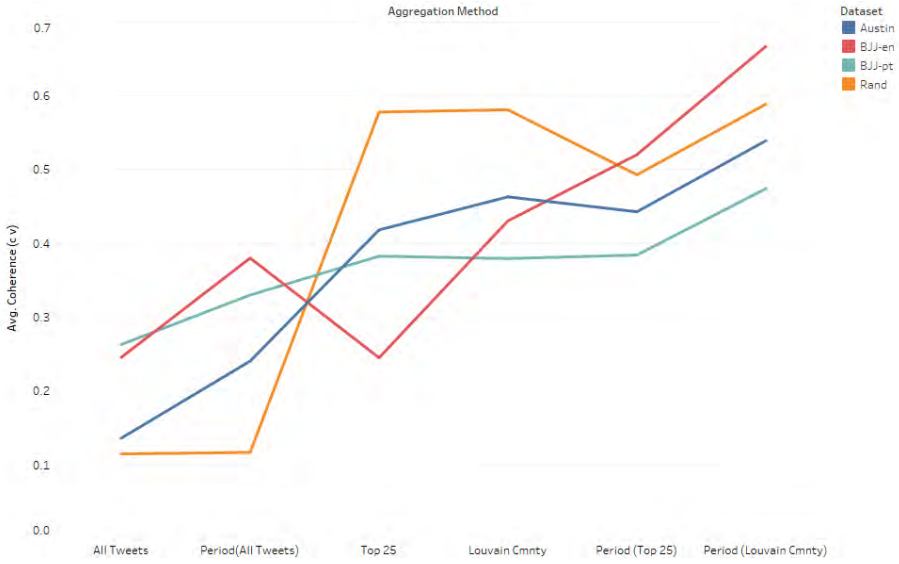

Figure 49: The average c_v coherence values for the different aggregation methods, evaluated using the hashtags connections network for the different datasets

Fig. 50, Fig. 51, and Fig. 52 visualize *Austin* dataset analysis. Figure 50 shows a comparison of topics found before and after the cleaning steps using the same input. Figure 51 shows three different topics found using only the LDA model without any aggregation techniques or additional visualization. And Figure 51 shows three sample visualizations for a topic discovered using the graph-based aggregation techniques. The words found in the LDA model without any additional aggregation are less specific and less descriptive as shown in Figure 51. But the topic found using the graph-based topic discovery method and the additional visualizations show a more clear and easy to interpret topic, as shown in Figure 51.

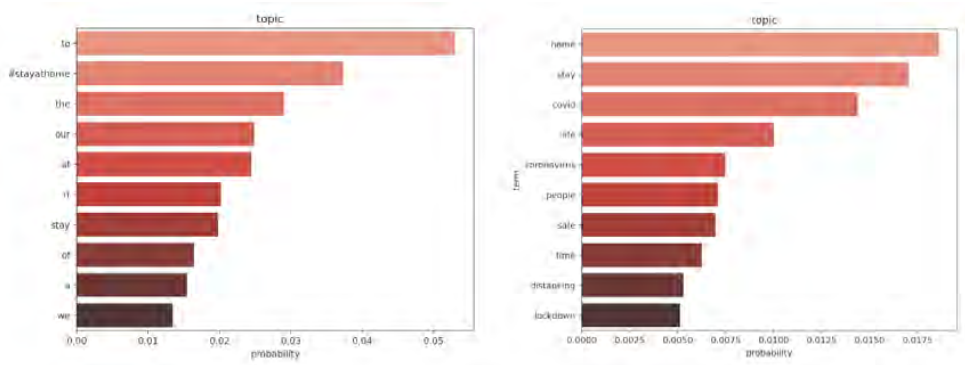

Figure 50: LDA model results before and after cleaning techniques. Initial result (left); result after cleaning techniques (right). Data extracted from a community in the *Covid* dataset.

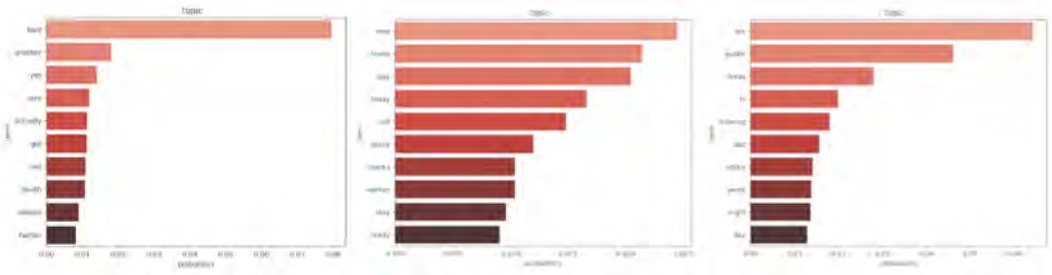

Figure 51: Three sample topics discovered using only the LDA model, without any aggregation techniques or additional visualizations. Topics extracted from the *Austin* dataset.

Figure 52: Three sample visualizations for a topic discovered using a graph-based aggregation technique. Topics extracted from a community found in the hashtags network in the *Austin* dataset. Figure 12 shows a larger version of the graph.

Pipeline

All experiments were conducted using the *pytwanalysis* package built for the thesis work, as outlined in Chapter VI: loading data into MongoDB, printing EDA, building graphs, identifying clusters and topics, and calculating metrics. Those experiments by themselves prove the performance and usability of the package. Additional experiments were done to show the execution times for loading data in the data management system to prove the performance improvements. Figure 53 and Table 32 show the execution time improvements for each collection after implementing the performance improvement methods described in Chapter V. The average execution time decreased by approximated 60%.

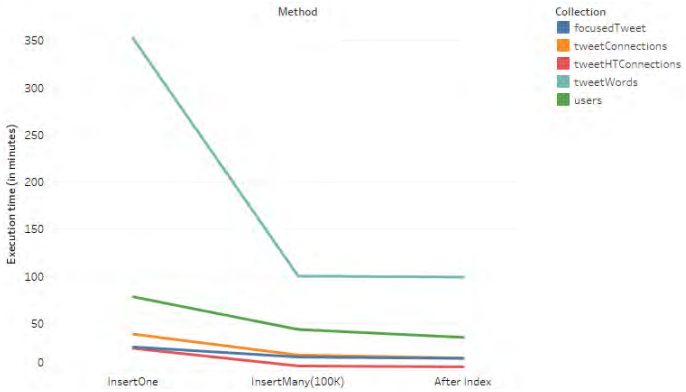

Figure 53: Execution time improvements for each collection after implementing the performance improvement methods

Table 32: Execution time of the different insert methods for different collections in the *MeToo* dataset

Collection	#of docs	Execution time (in minutes)			
		InsOne	InsM(100K)	After index	% decrease
focusedTweet	3,087,475	24.97	14.31	12.87	48.44%
tweetWords	65,664,493	352.47	100.23	99.12	71.87%
tweetConnections	6,920,655	38.64	16.36	13.16	65.94%
tweetHTConnections	6,155,025	23.52	4.81	3.88	83.484%
users	1,275,694	78.21	43.46	35.15	55.04%

VIII. SUMMARY AND CONCLUSIONS

This thesis work proposes a scalable way to gather, discover, analyze, and summarize large datasets from the Twitter social media platform. The system foundation is a suite of state-of-the-art techniques in social network analysis, and the final product is a scalable end-to-end data science pipeline that adapts to the dynamic and semi-structured nature of Twitter data.

The product of the thesis work is a released software package for general use that supports a data science pipeline for loading, storing, and analyzing Twitter data; a full NoSQL record-centric data management system for semi-structured dynamic data; a suite of improved and integrated techniques for multimodal graph creation, graph clustering, graph compression techniques, semantic analysis, and visualization tools to improve topic discovery.

The released python package *pytwanalysis* is available for installation via *pip* [1]. The package contains functionalities for acquiring data from Twitter's APIs [47, 46, 48], loading Twitter data from offline JSON files, storing and transforming the data in MongoDB, and printing exploratory data analysis. It also supports graph creation methods, graph clustering methods, graph compression for improved visualization, topic discovery methods, visualizations, and metrics calculation to evaluate clustering performance. The *pytwanalysis* package is designed as a modular, highly-customizable, and flexible software item that can adapt to the users' unique research question, dataset, and hardware specifications, as outlined in the package documentation [2]. Each method can be executed separately or in a series as an automation process option in the package. Each step of the data science pipeline can be customized in the automation process, but the default values for each method were set to optimize performance based on the findings of this thesis work.

We propose a data-management system that addresses the scalability with

number of records, being able to adapt to the dynamic nature of Twitter data, simplicity for end users, and support for data analytics. We outlined a record-centric data management system that stores the data and facilitates the different pieces of analysis, where MongoDB is used to store tweet records. This system addresses the shortcomings of MongoDB when retrieving aggregated data and joins by providing alternatives for data retrieval, and by organizing the data into multiple collections to improve data retrieval performance.

The multi-modal approach to creating networks using the Twitter data allows multi-purpose network analysis of the same dataset: connections based on retweets, mentions, replies, quotes, all types of user connections combined, and hashtag connections. Each of these types of networks can be used for a different purpose. Studying the different network types separately can be helpful for discovering different insights from the same data. The hashtags network was shown to have the highest connectivity compared to all other types of networks and was proven especially useful for helping with the topic analysis task.

Graph clustering implementation of *Spectral Clustering*, *Louvain*, and *TN-Neighborhoods* were evaluated with the existing metrics *Separability*, *Density*, *Average Clustering Coefficient*, and *computation time* on large Twitter datasets. None of these measures represented the true value of identifying communities that are highly connected through a few top degree vertices, so we introduced *Power Nodes Score* as a complementary measure. In conclusion, the *Spectral Clustering* method appears to have the most well separated and dense communities, the *TN-Neighborhoods* method the most well connected neighborhoods, and the *Louvain* method is a good general option that lives in between the other two methods. Even though the *Spectral Clustering* method outperforms the other methods in some categories, it lacks efficiency and it is not recommended to be used with large datasets. The *Spectral Clustering* algorithm is helpful for cases where we are

interested in how strong the relationship between vertices are. The *Louvain* method, on the other hand, which has better computation time, was created specifically for the task of finding communities within the networks, and can be used for networks with separate connected components, but it does not consider the strength of the vertices connectivity and focuses only on maximizing the modularity of the network. Both the *Spectral Clustering* and the *Louvain* methods find disjoint communities. In contrast, the *TN-Neighborhoods* method finds overlapping communities. These communities capture the connections around a set of high degree vertices. The *TN-Neighborhoods* method can be helpful for understanding the connections and discussions around particular influential vertices.

We have introduced graph reduction techniques to improve visualization. Twitter graphs are large, with highly varying vertex degrees and connectivity. Visualizations of large graphs are incomprehensible to the user, and we have proposed several automated context-aware approaches, depending on the application, to mitigate the issue. The simplification comes with a cost, as some information gets lost with the reduction, so each technique can be useful for answering different questions. The *Community Percentage* technique removes a percentage of the vertices of every community found in a given graph and will put emphasis on the vertices that are the centers of communities. The *Edge Weight* technique removes all edges with a weight less than a given number and will put emphasis on vertices that have stronger connections with other vertices. And the *Vertex Degree* technique removes all edges that connect two users with degree less than a given number and will put emphasis on the vertices with the highest degrees. The different techniques can be used by themselves or in combination.

For topic analysis, different aggregation methods were proposed to improve human interpretability of the topics. The topic coherence score was used as a guiding metric to evaluate different parts of aggregation, and the coherence was

proven to improve when using the proposed aggregation methods.

Implications and Future Work

The findings and product of this thesis can be used by a wide range of researchers that are interested in analyzing Twitter data for answering questions on several areas of study, from the social perspective to the computer science perspective, as our approach provides a pathway on analysis design, selection of parameters and algorithms, and it is all motivated by a research question. With the released package, it also provides a simple framework in charge of automating all steps of the analysis for those with less advanced technical skills. For those in Computer Science, it provides a framework for analysis that can be used as a starting point for further analysis so that the focus can be shifted to the task at hand instead of having to create an end-to-end system from scratch.

Even though the released package has functionalities to automate and make the analysis process simple, the users of the package are still required to have basic programming skills. No user interface was built and the release package was only created in Python. The package also only focuses on the data management, network analysis, and topic analysis. The Twitter analysis only focuses on three of the Twitter's APIs, but many more exist. The network analysis portion of the findings only focuses on undirected graphs, which stops the ability to analyse the connections in the network based on their direction. The graph reductions techniques, although very helpful for visualization, can result in some loss of information. The aggregation methods proposed for topic analysis did improve the topic discovery, but more efforts in making the topics more human interpretable can be made.

Future work could include the package being expanded with more functionality, such as sentiment analysis, fake news detection, and support for other Twitter APIs. A user interface can also be built to help users without programming skills. The

analysis can also be expanded to analyze directed graphs, and to construct similarity matrices that can focus on other aspects of the connectivity between vertices, other than just frequency. More types of networks can also be included in the analysis, such as the relationship between users based on location, or semantic networks using the relationships between words. More efforts can also be made to improve human interpreatability in the topics, and more algorithms for topic discovery can be evaluated.

APPENDIX SECTION

APPENDIX A: Graph Analysis

Appendix A contains larger versions of the graph visualizations shown in chapter III.

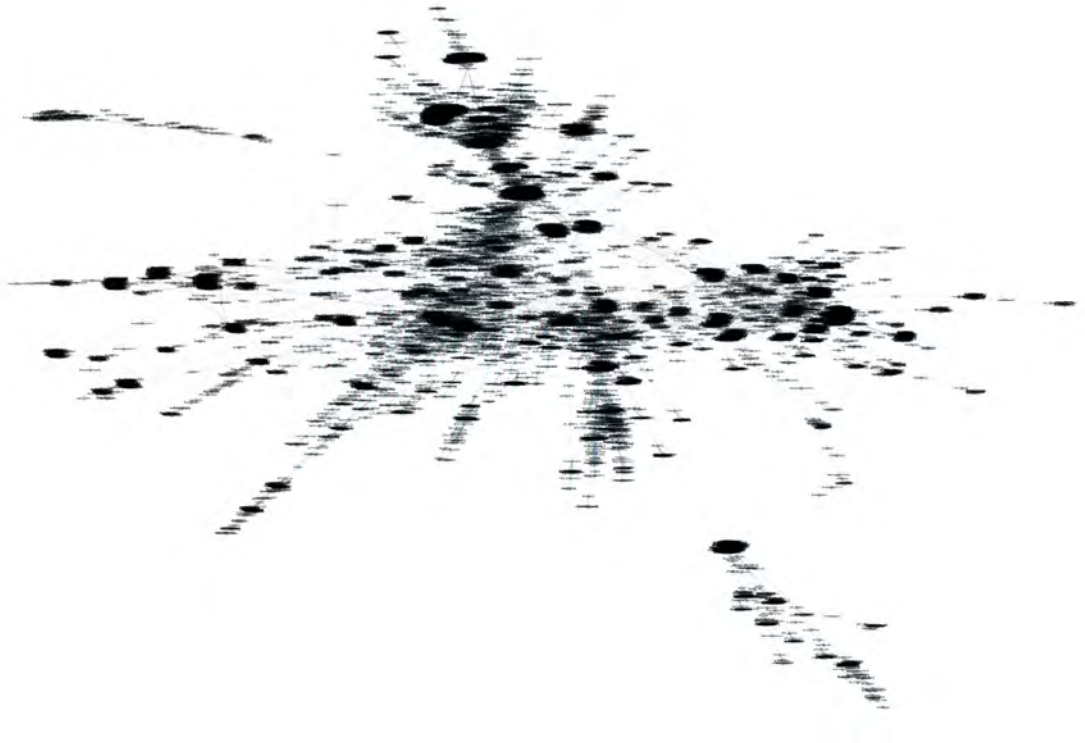

Figure A.1: Graph extracted from the *Austin* dataset with a total of 10,448 vertices and 11,977 edges, with no reduction

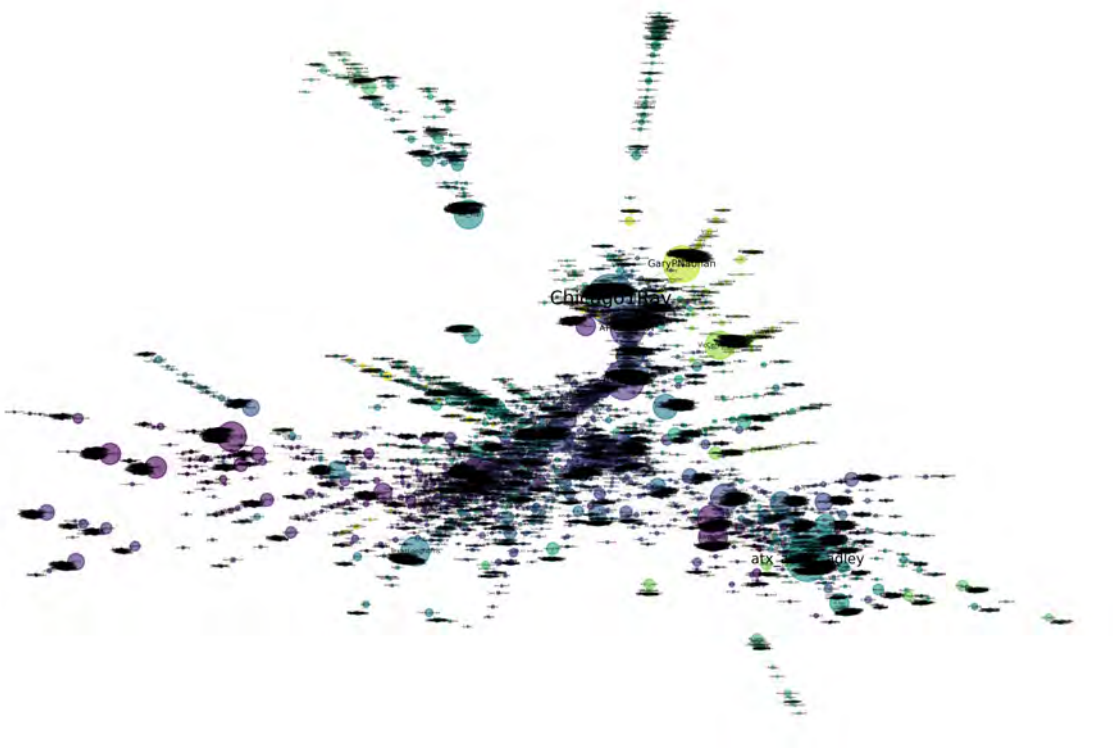

Figure A.2: Same graph data as in Figure A.1, but with the size of the vertices scaled based on their degree

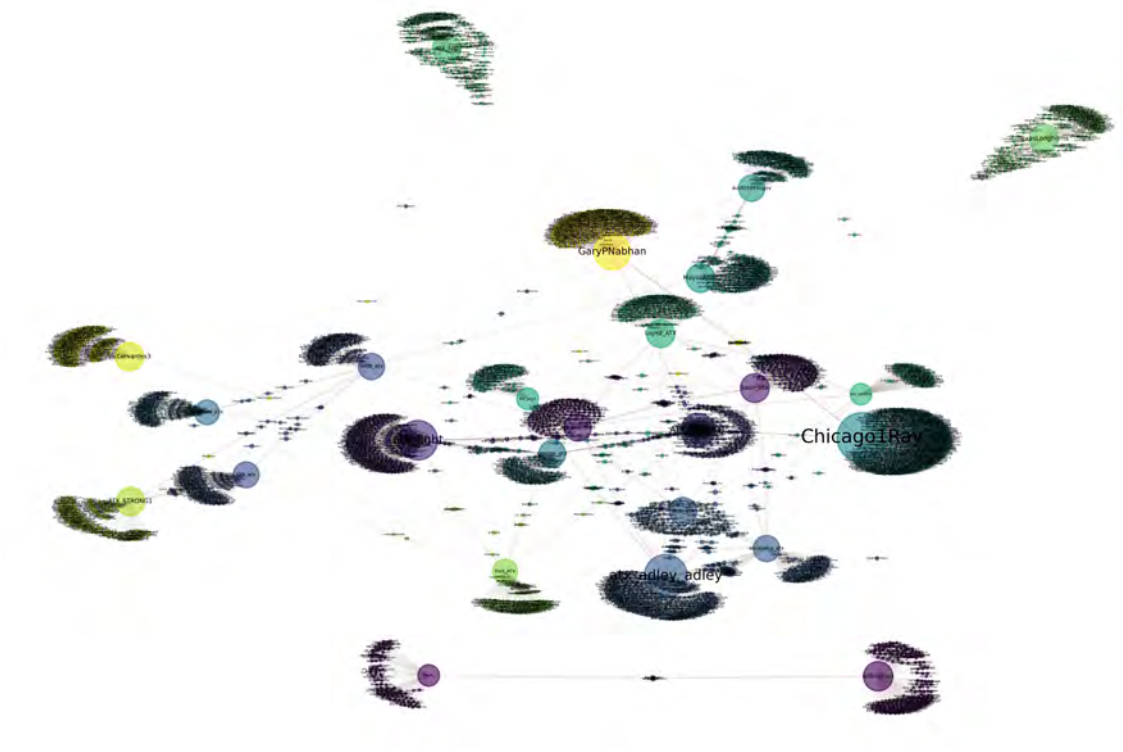

Figure A.3: Same graph data as in Figure A.1, but using the *Vertex Degree* reduction technique by removing every edge connecting two vertices with degree less than 115

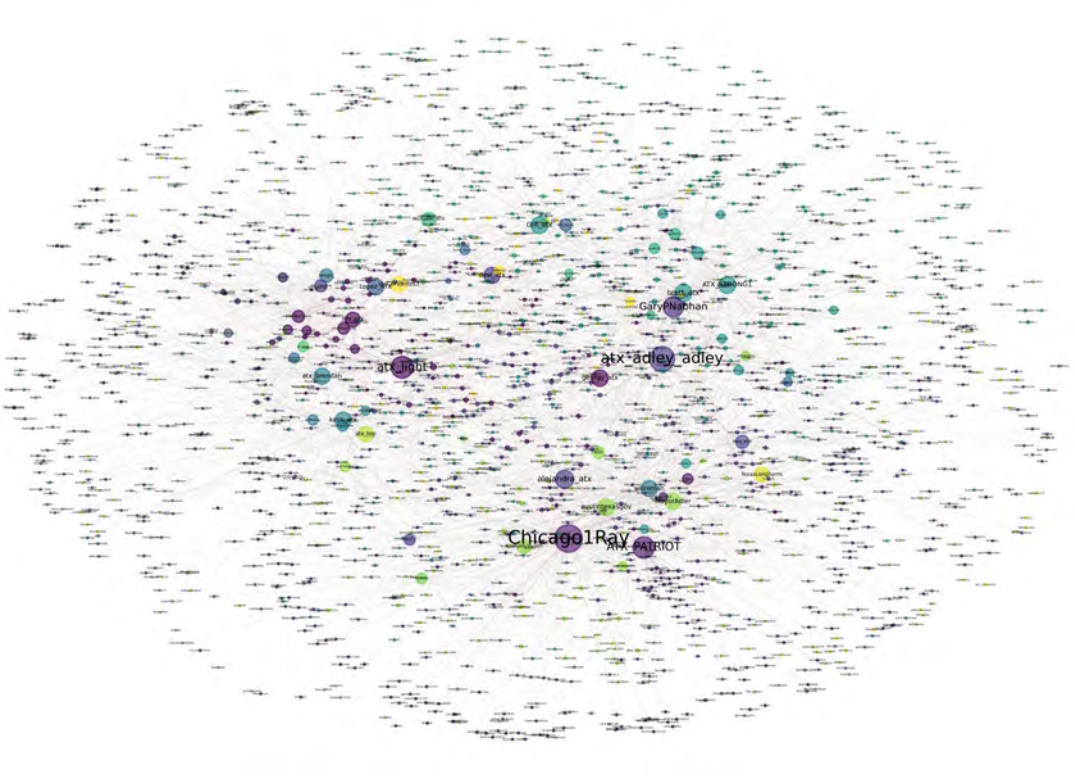

Figure A.4: Same graph data as in Figure A.1, but using both the *Community Percentage* and the *Vertex Degree* reduction techniques, with every community found in the graph reduced by 70%, and then every edge from the resulting graph with degree less than 25 removed

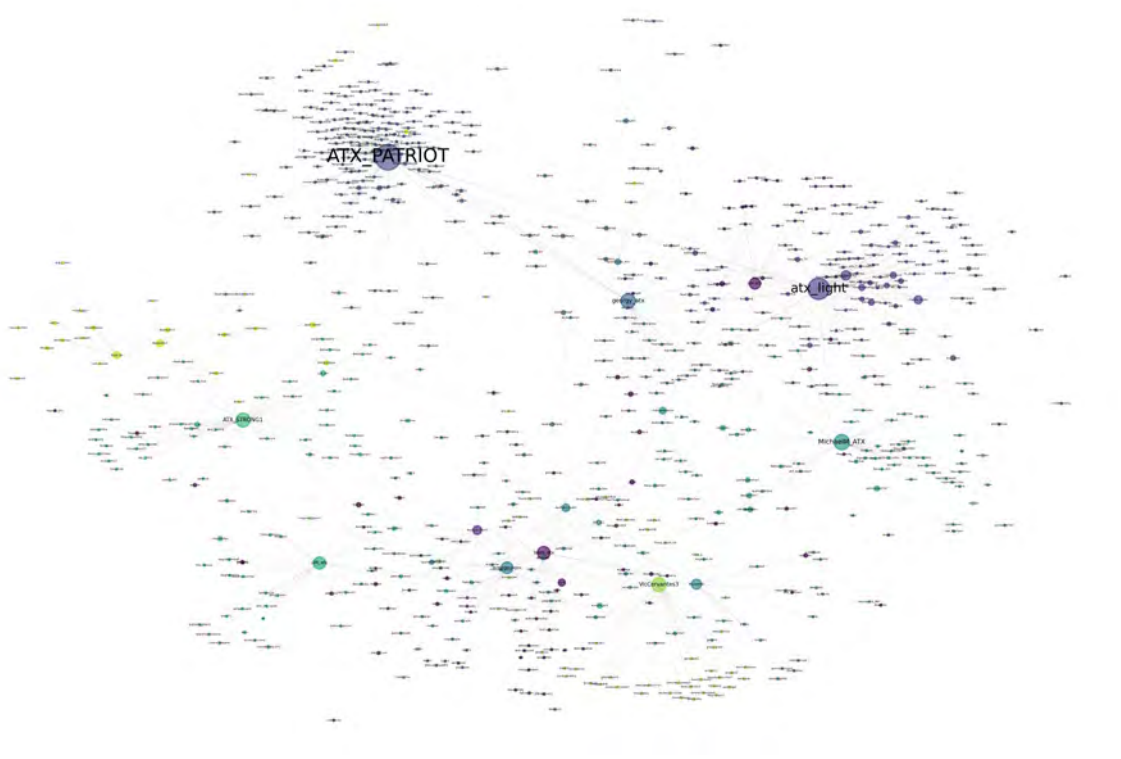

Figure A.5: Same graph data as in Figure A.1, but using both the *Edge Weight* and the *Community Percentage* reduction techniques, with every edge with weight 1 removed, then from the resulting graph the communities are reduced by 2%

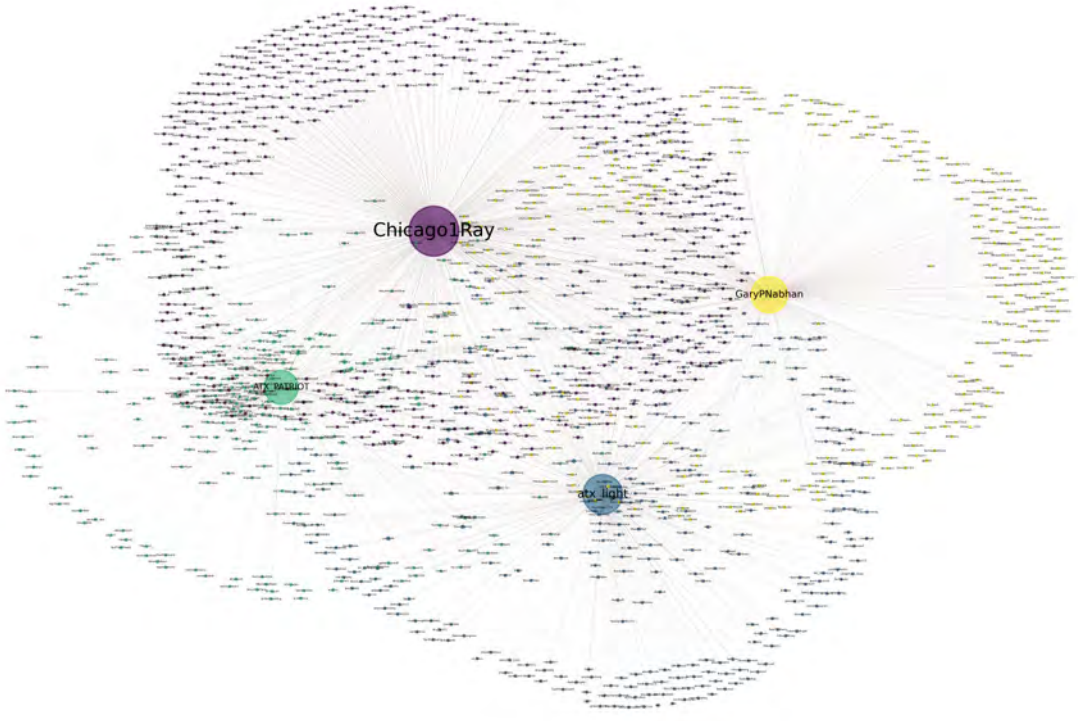

Figure A.6: Same graph data as in Figure A.1, but using the *Vertex Degree* reduction technique by removing every edge connecting two vertices with degree less than 355

APPENDIX B: Data Management

Appendix B contains additional details about the data dictionary of the data management system introduced in chapter V.

Table B.1: Description of the fields available on the *tweetConnections* collection

tweetConnections	
This collection stores the edges that connect two tweets together, either by retweets, quotes, replies, or mentions. It contains information about the tweet where the connection came from, and about the two users that connected. This collection is later used to build the edges for the graph analysis.	
Field	Description
tweet_id_str	The Id of the tweet that has this connection.
type_of_connection	The type of connection. *(retweet, quote, reply, or mention.)*
user_id_str_a	The user Id of the user that initiated the connection. For example, the user that replied to another user.
screen_name_a	The screen name of the user that initiated the connection. For example, the user that replied to another user.
user_id_str_b	The user Id of the user that received the connection. For example, another user retweets their tweet.
screen_name_b	The user screen name of the user that received the connection. For example, another user retweets their tweet.
desc	Describes the type of connection. Example of a possible value would be *user a mentioned user b*.
retweeted_status_id	The Id of the original tweet in case of retweets.
quoted_status_id	The Id of the original tweet in case of quotes.
in_reply_to_status_id	The Id of the original tweet in case of replies.
edge_screen_name_directed_key	A key created to help with directed graph queries. It basically concatenates the user_a and user_b screen names.
edge_screen_name_undirected_key	A key created to help with undirected graph queries. It concatenates the user_a and user_b screen names in alphabetical order.
tweet_created_at	The timestamp of the tweet.
tweet_seq_no	The seq_no of the tweet that created this connection.
is_bot	Identifies if this connection is for a bot or not. For example, this will be set to 1 if either user a or b is a bot. This field is not automatically calculated. A text file with the list of Ids that are bots can be used to update this field.

Table B.2: Description of the fields available on the *tweetHTConnections* collection

tweetHTConnections	
This collection stores the edges that connect two hashtags together. If two hashtags were used in the same tweet, a record will be created in this collection. It contains information about the tweet where the connection happened, and about the two hashtags. This collection is later used to build the edges for the graph analysis.	
Field	Description
tweet_id_str	The Id of the tweet that has this connection.
ht_a	The first hashtag.
ht_b	The second hashtag.
ht_key	A key useful for graph queries. It concatenates ht_a and ht_b.
tweet_created_at	The timestamp of the tweet that created this connection.
tweet_seq_no	The seq_no of the tweet that has this connection.
is_bot	Identifies if this connection is for a tweet created by a bot or not.

Table B.3: Description of the fields available on the *tweetWords* collection

tweetWords	
This collection stores each word separately for every tweet, and some additional information about the tweet. It also includes interesting tags about that word. (e.g English or not, verb or not, etc.)	
Field	Description
word	The original word.
word_tag	It identifies the type of word. ADJ, for adjectives, VERB, for verbs, NOUN, for nouns, and ADV for adverbs.
word_lemm	The word after lemmatization.
word_syl	The word broken down into syllables.
stop_word_fl	It identifies if the word is a stop word or not.
en_word_fl	It identifies if the word is in English or not.
tweet_id_str	The tweet Id where that word came from.
text	The tweet text where that word came from.
user_id	The user Id of the tweet where that word came from.
tweet_created_at	The timestamp of the tweet where that word came from.
tweet_seq_no	The seq_no of the tweet where that word came from.
seq_no	The word seq_no.
is_bot	Same as in the *focusedTweet* collection.
is_bot_connection	Same as in the *focusedTweet* collection.

Table B.4: Description of the fields available on the *focusedTweet* collection

focusedTweet	
This collection stores only the most interesting information about the tweet. The definition of *interesting* can be different depending on the research question, so settings can be updated to drive what fields are interesting. Some core fields will always be available no matter the settings.	
Field	Description
id_str	The tweet unique Id. This will be a key field that will be used to connect other collections together.
text	The tweet text.
text_lower	The tweet text with all lower case.
quote_text	The text of the quoted tweet, in case there is a quote associated to the tweet.
retweeted_text	The original tweet text in case of retweets.
text_combined	Tweet text combining the tweet text, original text, and quoted text.
text_combined_clean	Same as text_combined, but after going through a series of cleaning steps.
year	The year the tweet was created.
month_name	The name of the month that the tweet was created.
month_no	The month the tweet was created.
day	The day the tweet was created.
user_id	The user Id of the user that created the tweet.
hashtags	A list of all hashtags that were included in the tweet.
created_at	The timestamp the tweet was created.
lang	The language used in the tweet text.
in_reply_to_status_id_str	In case of replies, this field will show the Id of the tweet that the person was replying to.
in_reply_to_screen_name	In case of replies, this field will show the screen name of the user of the tweet that the person was replying to.
user_name	The user name of the person that created the tweet.
user_screen_name	The screen name of the person that created the tweet.
Added fields	
seq_no	Field created to uniquely identify each of the tweets. This field is used in the recovery process to identify the tweets that have already been processed.
is_bot	Identifies if the user that created the tweet is a bot or not. This field is not automatically calculated. A text file with the list of Ids that are bots can be used to update this field.
is_bot_connection	Identifies if the tweet is part of an edge between users that are bots. For example, this will be set to 1 if the tweet was a reply or a retweet to a user that is a bot. This field is not automatically calculated. A text file with the list of Ids that are bots can be used to update this field.

Table B.5: Description of the fields available on the *users* collection

users
This collection stores interesting information about the tweet user. Some core fields will always exist, but similar to the *focusedTweet* collection, settings are available to drive what fields are considered *interesting*. The same user can appear multiple times in a dataset with different values; for example, the same user can exist with two separate descriptions. This collection will not store multiple records for the same user and it will have only the first values found for each user. Other records for the same user will be ignored.

Field	Description
screen_name	The screen name of the user.
user_id	The unique Id for the user that was generated in Twitter.
name	The name of the user.
user_created_at	The date and time the user was created in Twitter.
location	The description the user used to identify their location.
location_clean	Same as *location*, but cleaning the special characters.
description	The description the users decided to give themselves.
description_clean	Same as *description*, but cleaning the special characters.
user_type	The method used to extract the information about this user. Twitter's documents have user information in the main document, but also under the information about the original tweet in case of retweets and quotes. This field will identify where the user was extracted from. The values available will be *tweet*, *retweet*, *quote*, *reply*, and *mention*.
is_bot	Identifies if the user is a bot or not. This field is not automatically calculated. A text file with the list of Ids that are bots can be used to update this field.

APPENDIX C: Pipeline

Appendix C contains additional details about the methods of the *pytwanalysis* package introduced in chapter VI.

Table C.1: Parameters expected on the *search7dayapi* and *searchPremiumAPI* methods

Parameter	Description	Used in method
consumer_key	User's consumer key.	search7dayapi
consumer_secret	User's consumer secret.	search7dayapi
access_token	User's access token.	search7dayapi
access_token_secret	User's access token secret.	search7dayapi
result_type	Options: *recent*, *popular*, or *mixed*.	search7dayapi
lang	Language to filter the tweets.	search7dayapi
query	The query that will be used to filter the tweets.	search7dayapi searchPremiumAPI
max_count	The number of tweets to be returned at a time.	search7dayapi searchPremiumAPI
twitter_bearer	Bearer authentication token created from the *consumer_key* and *consumer_secret*.	searchPremiumAPI
api_name	The options are either *30day* or *FullArchive*.	searchPremiumAPI
dev_environment	The name of the environment created on the Twitter developer's account.	searchPremiumAPI
date_start	The start date that will be used to filter the tweets.	searchPremiumAPI
date_end	The end date that will be used to filter the tweets.	searchPremiumAPI
next_token	Then token that points to the previous search done with the same query.	searchPremiumAPI

Table C.2: Description of the output metrics printed as part of the *eda_ analysis* method

Metric	Description
Tweet counts	The number of tweet documents in the database, divided by the following categories: **(i)** Total Original Tweets, **(ii)** Total Replies, **(iii)** Total Retweets, and **(iv)** Total Tweets.
Tweet counts by language	The number of tweet documents for each language used in the tweets.
Tweet counts by month	The number of tweet documents for each month/year.
Tweet counts by file	The number of tweet documents imported from each of the json files.
User counts	The number of users in the database, divided by the following categories: **(i)** tweet: Users with at least one document in the database, **(ii)** retweet: Users that were retweeted, but are not part of previous group, **(iii)** quote: Users that were quoted, but are not part of previous groups, **(iv)** reply: Users that were replied to, but are not part of previous groups, and **(v)** mention: Users that were mentioned, but are not part of previous groups.
All User Connections Graph	The metrics for the graph created based on the users connecting by retweets, quotes, mentions, and replies. The following graph metrics will be printed: **(i)** # of Vertices, **(ii)** # of Edges, **(iii)** # of Vertices of the largest connected components, **(iv)** # of Edges of the largest connected components, **(v)** # of Disconnected Graphs: The number of sub-graphs within the main graph that are not connected to each other, **(vi)** # of Louvain Communities found in the largest connected component, **(vi)** Degree of the top 5 most connected users, **(vii)** Average Vertex Degree of largest connected graph, **(viii)** Plot of the Louvain community distribution, **(ix)** Disconnected graphs distribution: A plot of a graph showing the distribution of the disconnected graphs. It shows the total number of vertices and edges for each of the disconnected graphs.
Mentions User Connections Graph	The same metrics as the *All User Connections* graph, but only considering the connections made by mentions.
Retweets User Connections Graph	The same metrics as the *All User Connections* graph, but only considering the connections made by retweets.
Replies User Connections Graph	The same metrics as the *All User Connections* graph, but only considering the connections made by replies.
HT Connection Graph	The same metrics as the *All User Connections* graph, but only considering the connections made by hashtags.

Table C.3: Description of the parameters used on *plotSpringLayoutGraph* method

Parameter	Description	Default
G	The graph to plot.	
v_graph_name	The name of the file to be saved.	
v_scale	Scale factor for positions.	
v_k	Optimal distance between vertices.	
v_iterations	Number of iterations of spring-force relaxation.	
cluster_fl	Options are Y or N. It identifies if the clusters should be identified in the plot with different colors.	
v_labels	In case parameter cluster_fl is 'Y', then this parameter expects the labels of vertices to identify the communities.	
v_node_color	A hexadecimal value representing a color to be used for the vertex color.	#A0CBE2
v_edge_color	A hexadecimal value representing a color to be used for the edge color.	#A0CBE2
v_width	The width of the edge lines.	0.05
v_node_size	The size of the vertex.	0.6
v_font_size	The font size for the vertices.	0.4
v_dpi	The dpi used to create the image file.	900
v_alpha	The transparency of the vertex.	0.6
v_linewidths	The line width of the vertex border.	
scale_node_size_fl	A flag to identify if the size of the vertices should be scaled based on their degree or not. Options are 'Y' or 'N'.	'Y'
node_size_multiplier	A multiplier number used to scale the size of the vertices. This option is only applicable in case the *scale_node_size_fl* is set to 'Y'.	6
font_size_multiplier	A multiplier number used to scale the size of the labels. This option is only applicable in case the *scale_node_size_fl* is set to 'Y'.	7
replace_existing_file	In case a file with the same name already exist in the given path, the flag will identify if the file should be replaced or not. Options are 'Y' or 'N'.	Y

Table C.4: Description of the core parameters in the *setConfigs* method.

Parameter	Description
type_of_graph	This setting defines the type of graph to analyze. Six different options are available: 'user_conn_all', 'user_conn_retweet', 'user_conn_quote', 'user_conn_reply', 'user_conn_mention', and 'ht_conn'.
period_arr	An array of start and end dates can be set so that the pipeline creates a separate analysis folder for each of the time periods in the array.
create_top_nodes_files_flag	If this setting is set to 'Y', the pipeline will create separate analysis folders for all the top degree vertices.
top_degree_start/end	If the setting *create_top_nodes_files_flag* was set to 'Y', then these numbers will define how many top degree vertex sub-folders to create.
period_top_degree_start/end	If the setting *create_top_nodes_files_flag* was set to 'Y', then these numbers will define how many top degree vertex sub-folders for each period to create.
create_nodes_edges_files_flag	If this setting is set to 'Y', the pipeline will create two files for each graph and sub-graph. One file with the edge list, and one with the vertex list and their respective degree.
create_graphs_files_flag	If this setting is set to 'Y', the pipeline will plot the graph showing all the connections.
create_topic_model_files_flag	If this setting is set to 'Y', the pipeline will create topic discovery related files for each folder. It will create a text file with all the tweets that are part of that folder. It will also train a LDA model based on the tweets' texts and plot a graph with the results.
create_ht_frequency_files_flag	If this setting is set to 'Y', the pipeline will create hashtag frequency files for each folder. It will create a text file with the full list of hashtags and their frequency, a wordcloud and a barchart showing the most frequently used hashtags.
create_words_frequency_files_flag	If this setting is set to 'Y', the pipeline will create word frequency files for each folder. It will create a text file with a list of words and their frequency, and a wordcloud and a barchart showing the most frequently used words.
create_ht_conn_files_flag	If this setting is set to 'Y', the pipeline will plot hashtag connections graphs. This can be used when user connections are being analyzed, but it could still be interesting to see the hashtags connections made by that group of users.
create_timeseries_files_flag	If this setting is set to 'Y', the pipeline will create time-series graphs for each folder representing the tweet count by day, and the top hashtags frequency count by day.
create_community_files_flag	If this setting is set to 'Y', the pipeline will use the Louvain method to assign each vertex to a community. A separate folder for each of the communities will be created with all the analysis files.

Table C.5: Description of the parameters in the *setConfigs* method that complement and give more details to the core parameters

Parameter	Description
num_of_topics	If the setting *create_topic_model_files_flag* was set to 'Y', then this number will be used to send as input to the LDA model. If no number is given, the pipeline will use 4 as the default value.
top_no_word_filter	If the setting *create_words_frequency_files_flag* was set to 'Y', then this number will be used to decide how many words will be saved in the word frequency list text file. If no number is given, the pipeline will use 5000 as the default value.
top_ht_to_ignore	If the setting *create_ht_conn_files_flag* was set to 'Y', then this number will be used to choose how many top hashtags can be ignored. Sometimes ignoring the main hashtag can be helpful in visualizations to discover other interesting structures within the graph.
commty_edge_size_cutoff	If the setting *create_community_files_flag* was set to 'Y', then this number will be used as the community size cutoff number. Any communities that have less vertices than this number will be ignored. If no number is given, the pipeline will use 200 as the default value.

Table C.6: Description of the parameters in the *setConfigs* method that are related to the graphs' visualizations files

Parameter	Description
create_graph_without_node_scale_flag	For each graph created, if this setting is set to 'Y', the pipeline will try to plot the full graph with no reduction and no logic for scaling the vertex size.
create_graph_with_node_scale_flag	For each graph created, if this setting is set to 'Y', the pipeline will try to plot the full graph with no reduction, but with additional logic for scaling the vertex size.
graph_plot_cutoff_no_nodes graph_plot_cutoff_no_edges	For each graph created, these numbers will be used as cutoff values to decide if a graph is too large to be plotted or not. Choosing a large number can result in having the graph take a long time to run. Choosing a small number can result in graphs that are too reduced and with little value or even graphs that can't be printed at all because they can't be reduced further.

Table C.7: Description of the parameters in the *setConfigs* method that are related to the graphs' reduction settings

Parameter	Description
create_reduced_graph_flag	For each graph created, if this setting is set to 'Y', the pipeline will try to plot the reduced form of the graph.
reduced_graph_comty_contract_per	If the setting *create_reduced_graph_flag* was set to 'Y', then this number will be used to reduce the graphs by removing a percentage of each community found in that particular graph. The logic can be run multiple times with different percentages. For each time, a new graph file will be saved with a different name according to the parameter given.
reduced_graph_remove_edge_weight	If the setting *create_reduced_graph_flag* was set to 'Y', then this number will be used to reduce the graphs by removing edges that have weights smaller than this number. The logic can be run multiple times with different percentages. For each time, a new graph file will be saved with a different name according to the parameter given.
reduced_graph_remove_edges	If this setting is set to 'Y', and the setting *create_reduced_graph_flag* was set to 'Y', then the pipeline will continuously try to reduce the graphs by removing edges of vertices with degrees smaller than this number. It will stop the graph reduction once it hits the the values set in the *graph_plot_cutoff parameters*.

Table C.8: Description of the possible files that can get created as output of output of method *edge_files_analysis* as part of the analysis automation process

File name	Description
G_Measures-(All).txt	Some metrics of the graph. (e.g. total number of vertices and edges.)
G_Measures-(LargestCC).txt	Some metrics of the largest connected component of the graph. (e.g. total number of vertices and edges.)
G_Edges.txt	List of edges and their respective weight.
G_NodesWithDegree.txt	List of all vertices with their respective degrees.
G_Nodes_WordCloud.png	Word cloud representing the vertices in the graph, weighted by the vertex's degree.
G_Graph.png	Plot of the full graph without any reduction. The size of the vertices are scaled based on the vertex's degree.
G_Graph(WithoutScale).png	Plot of the full graph without any reduction and without any scale for the vertex size.
G_Graph-(ReducedGraph)[%Parameters].png	Plot of the graph after reduction. The parameters used for reduction will appear in the file name inside brackets.
T_tweetTextsForTopics.txt	Tweet texts excluding some special characters, stop words, hashtags, and mentions.
Topics-(LDA model).png	Topic discovery plot using LDA model.
T_HT_FrequencyList.txt	A list of all hashtags and the number of times they were used.
T_HT_Top30_BarChart.png	A Bar Chart showing the top 30 hashtags.
T_HT_Top30_BarChart-(Excluding Top1).png	A Bar Chart showing the top hashtags, excluding the top 1.
T_HT_Top30_BarChart-(Excluding Top2).png	A Bar Chart showing the top hashtags, excluding the top 2.
T_HT_WordCloud.png	Word cloud representing the hashtags used, weighted by their frequency.
T_Words_FrequencyList.txt	A list of the top words used and the number of times they were used.
T_Words_Top30_BarChart.png	A Bar Chart showing the top 30 words.
T_Words_WordCloud.png	Word cloud representing the words used, weighted by their frequency.
TS_TweetCount.png	Timeseries graph showing the tweet count by day.
TS_TweetCountByHT[1-5]	Timeseries graph showing the hash count by day of the the top 5 hashtags.
ht_edges.txt	List of hashtag edges created based on the connections made by two hashtags being used on the same tweet. This file is used to create the hashtag connections graph.
HTG_G_Graph.png	Hashtag connection graph.
HTG_G_Graph-(ReducedGraph)[%Parameters].png	Reduced hashtag connection graph. The parameters used for reduction will appear in the file name inside brackets.

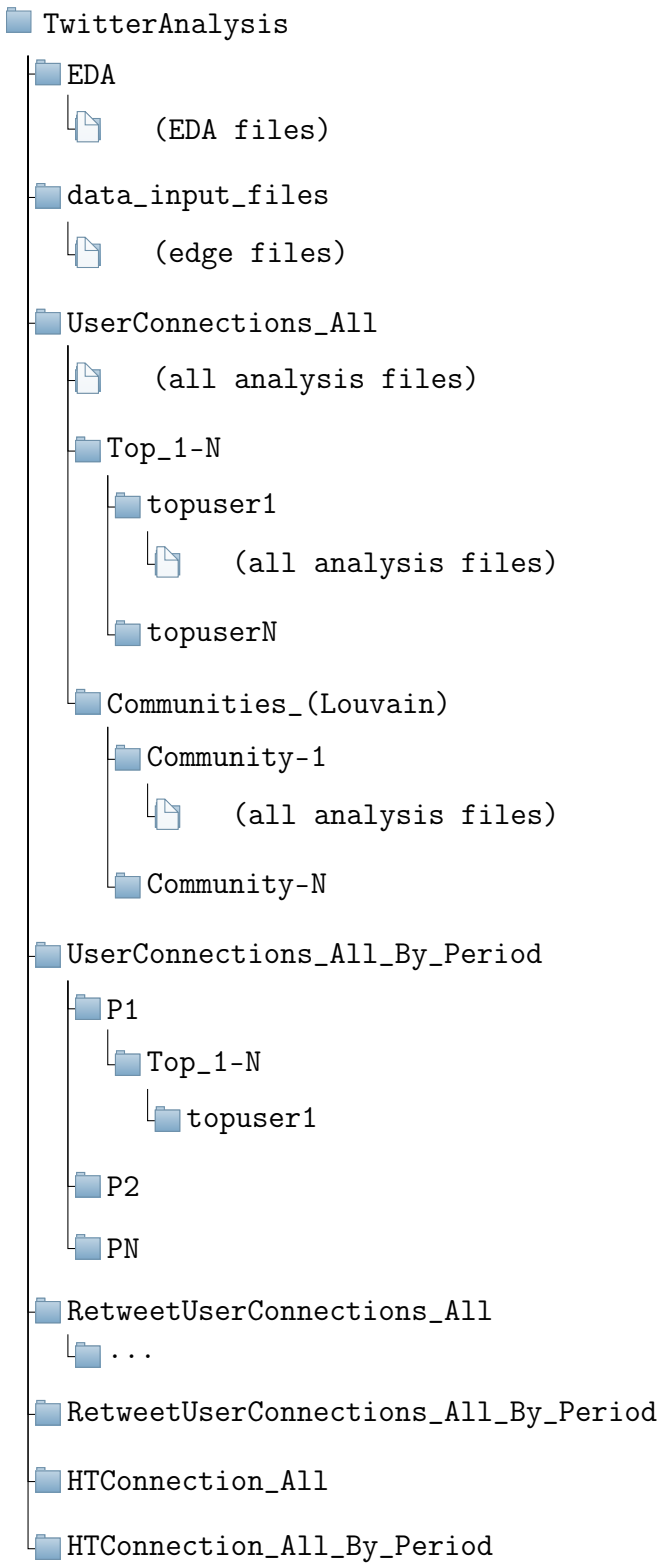

Figure C.1: Sample folder structure created as part of output of the analysis automation process

REFERENCES

[1] "pytwanalysis package." `https://pypi.org/project/pytwanalysis/`.

[2] "pytwanalysis documentation." `https://github.com/lianogueira/pytwanalysis-documentation/`.

[3] S. A. Williams, M. M. Terras, and C. Warwick, "What do people study when they study twitter? classifying twitter related academic papers," *Journal of Documentation*, 2013.

[4] A. Bruns, K. Weller, M. Zimmer, and N. J. Proferes, "A topology of twitter research: disciplines, methods, and ethics," *Aslib Journal of Information Management*, 2014.

[5] A. Signorini, A. M. Segre, and P. M. Polgreen, "The use of twitter to track levels of disease activity and public concern in the us during the influenza a h1n1 pandemic," *PloS one*, vol. 6, no. 5, p. e19467, 2011.

[6] W. Pearce, K. Holmberg, I. Hellsten, and B. Nerlich, "Climate change on twitter: Topics, communities and conversations about the 2013 ipcc working group 1 report," *PloS one*, vol. 9, no. 4, p. e94785, 2014.

[7] J. Bollen, H. Mao, and X. Zeng, "Twitter mood predicts the stock market," *Journal of computational science*, vol. 2, no. 1, pp. 1–8, 2011.

[8] A. Bovet and H. A. Makse, "Influence of fake news in twitter during the 2016 us presidential election," *Nature communications*, vol. 10, no. 1, pp. 1–14, 2019.

[9] W. Ahmed, J. Vidal-Alaball, J. Downing, and F. L. Seguí, "Covid-19 and the 5g conspiracy theory: social network analysis of twitter data," *Journal of Medical Internet Research*, vol. 22, no. 5, p. e19458, 2020.

[10] H. Sha, M. A. Hasan, G. Mohler, and P. J. Brantingham, "Dynamic topic modeling of the covid-19 twitter narrative among us governors and cabinet executives," *arXiv preprint arXiv:2004.11692*, 2020.

[11] L. Zou, N. S. Lam, S. Shams, H. Cai, M. A. Meyer, S. Yang, K. Lee, S.-J. Park, and M. A. Reams, "Social and geographical disparities in twitter use during hurricane harvey," *International Journal of Digital Earth*, vol. 12, no. 11, pp. 1300–1318, 2019.

[12] L. Sinnenberg, A. M. Buttenheim, K. Padrez, C. Mancheno, L. Ungar, and R. M. Merchant, "Twitter as a tool for health research: a systematic review," *American journal of public health*, vol. 107, no. 1, pp. e1–e8, 2017.

[13] Y. Huang, D. Guo, A. Kasakoff, and J. Grieve, "Understanding us regional linguistic variation with twitter data analysis," *Computers, environment and urban systems*, vol. 59, pp. 244–255, 2016.

[14] A. Tumasjan, T. O. Sprenger, P. G. Sandner, and I. M. Welpe, "Predicting elections with twitter: What 140 characters reveal about political sentiment," in *Fourth international AAAI conference on weblogs and social media*, 2010.

[15] S. Bijarnia, P. Ilavarasan, and A. Kar, "Comparing servqual for transportation services in the sharing economy for emerging markets: Insights from twitter analytics," *Digital and Social Media Marketing. Advances in Theory and Practice of Emerging Markets.*

[16] D. Duran-Rodas, D. Villeneuve, and G. Wulfhorst, "Bike-sharing: the good, the bad, and the future -an analysis of the public discussion on twitter-," *European Journal of Transport and Infrastructure Research*, vol. 20, no. 4, pp. 38–58, 2020.

[17] O. Johnson, A. Hall-Phillips, T.-L. D. Chung, and H. Cho, "Are you connected through consumption? the role of hashtags in political consumption," *Social Media + Society*, vol. 5, no. 4, 2019.

[18] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas, "Comparing community structure identification," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, no. 09, p. P09008, 2005.

[19] S. E. Schaeffer, "Graph clustering," *Computer science review*, vol. 1, no. 1, pp. 27–64, 2007.

[20] J. Yang, J. McAuley, and J. Leskovec, "Detecting cohesive and 2-mode communities indirected and undirected networks," in *Proceedings of the 7th ACM international conference on Web search and data mining*, pp. 323–332, 2014.

[21] S. Kumar, W. L. Hamilton, J. Leskovec, and D. Jurafsky, "Community interaction and conflict on the web," in *World Wide Web*, pp. 933–943, 2018.

[22] M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi, "Measuring user influence in twitter: The million follower fallacy," in *fourth international AAAI conference on weblogs and social media*, 2010.

[23] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowledge and Information Systems*, vol. 42, no. 1, pp. 181–213, 2015.

[24] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[25] E. Jónsson and J. Stolee, "An evaluation of topic modelling techniques for twitter," 2015.

[26] R. Mehrotra, S. Sanner, W. Buntine, and L. Xie, "Improving lda topic models for microblogs via tweet pooling and automatic labeling," in *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pp. 889–892, 2013.

[27] H. Hromic, N. Prangnawarat, I. Hulpuş, M. Karnstedt, and C. Hayes, "Graph-based methods for clustering topics of interest in twitter," in *International Conference on Web Engineering*, pp. 701–704, Springer, 2015.

[28] M. Bastian, S. Heymann, M. Jacomy, *et al.*, "Gephi: an open source software for exploring and manipulating networks.," *Icwsm*, vol. 8, no. 2009, pp. 361–362, 2009.

[29] V. Batagelj and A. Mrvar, "Pajek-program for large network analysis," *Connections*, vol. 21, no. 2, pp. 47–57, 1998.

[30] M. Smith, N. Milic-Frayling, B. Shneiderman, E. Mendes Rodrigues, J. Leskovec, and C. Dunne, "Nodexl: a free and open network overview, discovery and exploration add-in for excel 2007/2010," 2010.

[31] J. Leskovec and R. Sosič, "Snap: A general-purpose network analysis and graph-mining library," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 8, no. 1, pp. 1–20, 2016.

[32] N. U. Rehman, S. Mansmann, A. Weiler, and M. H. Scholl, "Building a data warehouse for twitter stream exploration," in *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pp. 1341–1348, IEEE, 2012.

[33] M. R. Murazza and A. Nurwidyantoro, "Cassandra and sql database comparison for near real-time twitter data warehouse," in *2016 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, pp. 195–200, IEEE, 2016.

[34] M. B. Kraiem, J. Feki, K. Khrouf, F. Ravat, and O. Teste, "Modeling and olaping social media: the case of twitter," *Social Network Analysis and Mining*, vol. 5, no. 1, p. 47, 2015.

[35] D. Khanaferov, C. Luc, and T. Wang, "Social network data mining using natural language processing and density based clustering," in *2014 IEEE International Conference on Semantic Computing*, pp. 250–251, IEEE, 2014.

[36] U. Unturk, S. Ozcan, and H. GÃŒmÃŒskaya, "Tweetcuriosity: A simple preprocessing and analyzing tool for twitter data," in *International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering*, 12 2014.

[37] S. Saini and S. Kohli, "Healthcare data analysis using r and mongodb," in *Big Data Analytics* (V. B. Aggarwal, V. Bhatnagar, and D. K. Mishra, eds.), (Singapore), pp. 709–715, Springer Singapore, 2018.

[38] W. Wingerath, F. Gessert, and N. Ritter, "Twoogle: Searching twitter with mongodb queries," in *BTW 2019* (T. Grust, F. Naumann, A. Böhm, W. Lehner, T. HÃ€rder, E. Rahm, A. Heuer, M. Klettke, and H. Meyer, eds.), pp. 523–527, Gesellschaft fÃŒr Informatik, Bonn, 2019.

[39] D. T. Schroeder, K. Pogorelov, and J. Langguth, "Fact: a framework for analysis and capture of twitter graphs," in *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pp. 134–141, 2019.

[40] R. C. Van der Hulst, "Introduction to social network analysis (sna) as an investigative tool," *Trends in Organized Crime*, vol. 12, no. 2, pp. 101–121, 2009.

[41] C. R. Carter, L. M. Ellram, and W. Tate, "The use of social network analysis in logistics research," *Journal of Business Logistics*, vol. 28, no. 1, pp. 137–168, 2007.

[42] M. T. Heaney and S. D. McClurg, "Social networks and american politics: Introduction to the special issue," *American Politics Research*, vol. 37, no. 5, pp. 727–741, 2009.

[43] P. A. Rauschnabel, P. Sheldon, and E. Herzfeldt, "What motivates users to hashtag on social media?," *Psychology & Marketing*, vol. 36, no. 5, pp. 473–488, 2019.

[44] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." `http://snap.stanford.edu/data`, June 2014.

[45] Y. Liu, T. Safavi, A. Dighe, and D. Koutra, "Graph summarization methods and applications: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–34, 2018.

[46] "Standard 7day search api endpoint." `https://api.twitter.com/1.1/search/tweets.json`.

[47] "Premium 30day search api endpoint." `https://api.twitter.com/1.1/tweets/search/30day/`.

[48] "Premium full-archive search api endpoint." `https://api.twitter.com/1.1/tweets/search/fullarchive/`.

[49] "Account activity api endpoint." `https://api.twitter.com/1.1/account_activity/webhooks.json`.

[50] K. Giuffre, *Communities and networks: using social network analysis to rethink urban and community studies*. John Wiley & Sons, 2013.

[51] K. Crockett, D. Mclean, A. Latham, and N. Alnajran, "Cluster analysis of twitter data: a review of algorithms," in *Proceedings of the 9th International Conference on Agents and Artificial Intelligence*, vol. 2, pp. 239–249, Science and Technology Publications (SCITEPRESS)/Springer Books, 2017.

[52] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference* (G. Varoquaux, T. Vaught, and J. Millman, eds.), (Pasadena, CA USA), pp. 11 – 15, 2008.

[53] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[54] Y. Liu, T. Safavi, N. Shah, and D. Koutra, "Reducing large graphs to small supergraphs: a unified approach," *Social Network Analysis and Mining*, vol. 8, no. 1, p. 17, 2018.

[55] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[56] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[57] S. Wojcik and A. Hughes, "Sizing up twitter users." `https://www.pewresearch.org/internet/wp-content/uploads/sites/9/2019/04/twitter_opinions_4_18_final_clean.pdf`, 4 2019.

[58] M. R. Fellows, J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann, "Graph-based data clustering with overlaps," *Discrete Optimization*, vol. 8, no. 1, pp. 2 – 17, 2011. Parameterized Complexity of Discrete Optimization.

[59] G. Bello Orgaz, H. MenÃ©ndez, and D. Camacho, "Adaptive k-means algorithm for overlapped graph clustering," *International journal of neural systems*, vol. 22, p. 1250018, 06 2012.

[60] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Predicting positive and negative links in online social networks," in *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pp. 641–650, ACM, 2010.

[61] R. D. Alba, "A graph-theoretic definition of a sociometric clique," *Journal of Mathematical Sociology*, vol. 3, no. 1, pp. 113–126, 1973.

[62] D. Greene and P. Cunningham, "Producing a unified graph representation from multiple social network views," in *Proceedings of the 5th annual ACM web science conference*, pp. 118–121, 2013.

[63] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos, "Vog: Summarizing and understanding large graphs," in *Proceedings of the 2014 SIAM international conference on data mining*, pp. 91–99, SIAM, 2014.

[64] K. Sharma, S. Seo, C. Meng, S. Rambhatla, and Y. Liu, "Covid-19 on social media: Analyzing misinformation in twitter conversations," 2020.

[65] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, (Valletta, Malta), pp. 45–50, ELRA, May 2010. `http://is.muni.cz/publication/884893/en`.

[66] M. Röder, A. Both, and A. Hinneburg, "Exploring the space of topic coherence measures," in *Proceedings of the eighth ACM international conference on Web search and data mining*, pp. 399–408, 2015.

[67] J. R. Davenport and R. DeLine, "The readability of tweets and their geographic correlation with education," *arXiv preprint arXiv:1401.6058*, 2014.

[68] "The definitive list of social media acronyms and abbreviations, defined." `https://buffer.com/library/social-media-acronyms-abbreviations`.

[69] "Corpora of misspellings for download from the university of birkbeck." `https://www.dcs.bbk.ac.uk/~ROGER/corpora.html`.

[70] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[71] Z. Parker, S. Poe, and S. V. Vrbsky, "Comparing nosql mongodb to an sql db," in *Proceedings of the 51st ACM Southeast Conference*, pp. 1–6, 2013.

[72] C.-S. Atodiresei, A. Tănăselea, and A. Iftene, "Identifying fake news and fake users on twitter," *Procedia Computer Science*, vol. 126, pp. 451–461, 2018.

[73] G. Yadav, M. Joshi, and R. Sasikala, "Twitter data analysis: temporal and term frequency analysis with real-time event," in *IOP Conference Series: Materials Science and Engineering*, vol. 263, p. 042081, 2017.

[74] M. Trupthi, S. Pabboju, and G. Narasimha, "Sentiment analysis on twitter using streaming api," in *2017 IEEE 7th International Advance Computing Conference (IACC)*, pp. 915–919, IEEE, 2017.

[75] P. Kumar, "Analysis of a data processing pipeline for generating knowledge graphs from unstructured data: Data processing pipeline for knowledge graphs," Master's thesis, Delft University of Technology, 8 2020. EIT Digital Double Degree Programme, Epema, D.H.J. (mentor).

[76] A. Celesti, M. Fazio, and M. Villari, "A study on join operations in mongodb preserving collections data models for future internet applications," *Future Internet*, vol. 11, no. 4, p. 83, 2019.

[77] "Twitter data dictionary." `https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/overview/intro-to-tweet-json`.

[78] "Twitter developer." `https://developer.twitter.com/en`.

[79] "Search tweets api documentation." `https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/overview`.

[80] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx." `https://networkx.github.io/documentation/stable/reference/algorithms/index.html`, Jan 2008.

[81] "pytwanalysis source code." `https://github.com/lianogueira/pytwanalysis`.