CSC111 Final Project Report: #Partisanship

Jiajin Wu, Kenneth Miura, Tai Zhang

Tuesday, March 16, 2021

Problem Description and Research Question

Hashtags are a functional tag used in social media, indexing tweets and messages based on a specific set of characters, from #lovewins to #fakenews. In doing so, it tags the digital content to a specific topic, making it easily accessible to people looking for similar results. Twitter is a microblogging service, where users can push their messages or "tweets" to others in 140 character texts.

Popularized for the short but concise format, politicians are using Twitter as the loudspeaker to address the people. One example is the Trump presidency, where his tweets became polarizing areas of discussion, and in the last few weeks of his term, sparked controversy and incited violence onto the Capital of the United States. In the recent months, with the increase in political uprising around the world, as well as the continuation of the pandemic, hashtags such as "#trumpvirus" or "#AstraZeneca" trend, reflecting the topics on people's mind (Jiang et al.)

Within social media itself, we find that the use of hashtags tries to fit us into similarly minded groups known as echo chambers (Matuszewski and Szabó). In this algorithmic attempt to keep us interested in the media, it also inadvertently groups users together, isolating them into feeds with singular voices. So as the political divide grows, so does the isolation in social media.

Our group wanted to look into the politics behind the hashtags, and see if we can find the bias and motivations behind the keywords themselves.

Our question is: **Can we determine the political bias of a hashtag, and find similarly connected topics?**

By taking a critical view of the political social media landscape, and utilizing the graphs as well as the algorithmic processing of political tweets, we hope to create a project that better informs ourselves on the networks behind today's topics, from the hidden agendas, related topics that we may not have considered.

Dataset Description

This data was obtained from the website:

We are using two datasets hosted at `http://twitterpoliticians.org/download`, namely full_member_info.csv, which provides information about politicians, their parties, their political system and their political biases, and all_tweets_ids.csv (They have changed this file and split it up into three different years. On top of that they have added tweets from 2020 to 2021 after the phase2 of the project started), which provides tweets from these politicans from May 2017 to October 31st, 2019. Both of these datasets of 26 countries, but for the sake of this project, we restrict ourselves to US politicians, in order to look into an extremely polarized country with a relatively binary electoral system (University of Amsterdam).

Here is an example of data from full_member_info.csv:

m.name	m.member_id	m.party_id	m.uid	m.party
austin scott	101	0	234797704	Republican
glenn w. thompson	103	0	\N	Republican
robert e. latta	107	0	15394954	Republican
cathy mcmorris rodgers	110	0	17976923	Republican
k. michael conaway	114	0	295685416	Republican

There is more info in the dataset, but we only care about the name of the politician, and their political party, which is shown in the example.

Here is an example of data from all_tweets_ids.csv:

866224163207483392
866224414425219072
866224432515252225
866224455672049664
866224459396599809

We used a hydrator to hydrate all the tweet ids from the csv file into a jsonl format file that we could access the data (We decided not to show an example, because a single example has an immense amount of information, from the contents of the message, the information about the Tweeter as well as the the person tagged and any meta-data encapsulated). By "hydrating", we mean using tweet IDs to retrieve information about the tweet from Twitter. We use a hydrator that is on `https://github.com/DocNow/hydrator`. There are more information than we need in the jsonl file, so we wrote a python script called making_new_csv to transfer the hydrated all_tweets_ids (from `http://twitterpoliticians.org/download`) jsonl file to a new csv file with the information we desire. The new csv file consists of only american politicians, their partisan score(democrats' being 0, republicans' being 1), and the hashtags they posted. There are tweets with no hashtags, so for those tweets we will skip them, since they have no use towards our final goal. The way we store these hashtags are a set of strings for each politician. Here is an example of the csv file that is after our process:

name	partisan score	hashtags
Nancy Pelosi	0	{'FutureIsProgressive', 'ABetterDeal'}
Susan W. Brooks	1	{'Hoosiers', 'JakeStrong'}
Brenda Lawrence	0	{'BrusselsForum'}

The way we made the csv file is that we read full_member_info.csv(from `http://twitterpoliticians.org/download`) and accounts-twitter-data.csv (from `https://github.com/pablobarbera/twitter_ideology/blob/master/primary/data/accounts-twitter-data.csv`) and put the politicians' twitter user_ids and their partisan score into a dictionary (user_ids as the key and partisan score as the value). Then we use all these informations to find US politician tweets that have hashtags and collect the hashtags. Then we put the infomation into a csv format of name, partisan score, and their hashtags tweet by tweet. This csv file will then be accessed by another program and be turned into python graph data. The main reason we are doing this is there are over 10 million tweets in the all_tweets_ids file. The run time of our program will be really long. However, after using making_new_csv, we will have under 230,000 different tweets (we only use hashtags from those tweets) for our final data. One thing to be noticed when running making_new_csv is that if you try to run it using all_tweets_ids, then the run time will be longer than normal programs because of the size of all_tweets_ids.

Computational Plan

Our computational plan ended up being split across 2 different sections. The first section being the collection, filtering, and manipulation of the tweet data-set. The second section took the filtered tweet data and added it to a WeightedGraph, as well as weighting them and their respective edges.

Retrieving, filtering, and processing the tweets

As we mentioned earlier on in the data-set description, we utilized a publicly available hydrator, as well as a dataset of politically oriented tweets, as well as a database of the respective politicians, and their political affiliation. The module `making_new_csv` contains the main functions for transforming and filtering our 40GB hydrated.jsonl file of hydrated Twitter tweets into an easily accessible csv that can then be processed into a graph.

Specifically, our main function `get_us_hashtags` starts by calling `get_us_information`, a function that creates a dictionary where the key-value pairs are the politician's name, corresponding to their political aligning, where 0 represents a Democratic politician, and 1 represents a Republican politician. We used politicians since we could assume that their tweets matched their political orientation and party viewpoints. Our specific `full_member_info.csv` file ended up including both Senators, and House of Representatives, two large bodies of Congress that make up the majority of US political landscape, and provide the most influence for both national and local affairs. While we

attempted to include the 45th President Donald J. Trump in, we found that his tweets could not be hydrated due to a suspension of his Twitter account, something that we explain more in the discussion.

Through basic filters comparing specific rows of location, political stance, we built up a `us_politicians` dictionary that call when reading the main file of hydrated tweets. From there, it was a matter of checking if the tweet's author was in our dictionary, and passing the corresponding information of their political background, as well as the hashtags found within the tweet itself. We then write a corresponding row into the `total_filtered_politician.csv`.

Building our WeightedGraph

For the construction of the WeightedGraph, we have the `dataclasses.py` file, which contains the main data structures, as well as the `csv_to_graph.py` file that builds up the data structures using the filtered csv we created.

In our `dataclasses.py`, we have two main data structures. The first is a hashtag node which functions as a vertex. It contains information about the hashtag's use such as how many times it occurs in our data-set, whom it has been tweeted by, and its neighbours (other hashtags that have appeared in the same tweet). It also has algorithms for determining the weighting of the node itself, which represents the partisanship bias.

The second main data structure is our WeightedGraph, which represents the network of the hashtag vertexes. We chose a graph because it plays a central role in allowing traversal from one node to another, and also being able to compute the weighting of individual, and cumulative neighbours. Our WeightedGraph includes functions for mutating the data such as the dictionary of hashtag vertices, as well as computational algorithms for determining the weighting of an edge. For our WeightedGraph, the edge represents the similarity between two hashtags, calculated using two different methods that ultimately weigh their mutual occurrence.

In the `load_weighted_hashtags_graph`, I read each line in our filtered csv, and apply the computations when we add the vertex into the graph. Specifically, in the `add_vertex` function, if the hashtag item is not found in `self._vertices`, we add it to the graph with the corresponding weighting and party count values. However, if it's already in the graph, we call `.update()` on our vertex, which updates the count of a pre-existing vertex, as well as the partisanship. The partisanship bias on a single hashtag is calculated by:

$$
\begin{aligned}
&= \frac{\text{self.count\_rep} * \text{REPUBLICAN} + \text{self.count\_dem} * \text{DEMOCRATIC}}{\text{self.count}} \\
&= \frac{\text{self.count\_rep}}{\text{self.count}}
\end{aligned}
$$

Since the DEMOCRATIC counts equal 0 and REPUBLICAN counts equal 1, we simplify to the expression above. Thus, we have completed the first portion of the `load_weighted_hashtags_graph`, which builds up the graph's vertices, in this case hashtags, and calculates their individual node weighting.

Then, the function calls `add_edges`, which computes and builds the individual edges for the graph. Within `add_edges`, for each row of our filtered tweets, we check if there's more than two hashtags in a single tweet (e.g. #DACA, #DREAMers). If so, we add an edge to the graph for each unique pairing. Within the function for adding edges, we update the times the specific hashtag pairing has occurred, and update the edge weighting based on the variable given to `edge_format`.

The first edge weighting is called `edge_weight_absolute`. We take a fraction of the number of times it's been tweeted over half the number of it's individual occurences. For example, if #ProtectOurCare has been tweeted 5 times, and #KeepFamiliesTogether has been tweeted 4 times, all 4 times being with #ProtectOurCare, their weighting would be:

$$
\begin{aligned}
&= \frac{2 * 4(\text{occurrences together})}{5(\text{occurrences of \#ProtectOurCare}) + 4(\text{\#KeepFamiliesTogether})} \\
&= \frac{8}{9}
\end{aligned}
$$

Leading to this edge having a weight of 0.88. This algorithm for weighting is based on the fact that while two hashtags may occur very often together (#Trump, #ImpeachTrump), #Trump may occur very often elsewhere, leading to a less strongly correlation between the two hashtags. On the other hand, when we find #DACA and #Dreamers to occur most often together, they will result in a much stronger weighting. This method ultimately looks for "exclusive-pairing", highly weighting hashtags that occur most often together. This makes it very accurate for niche topics, but somewhat impractical for a general hashtag such as #Trump, who's weightings against other hashtags are extremely low.

The other method of weighting is called `edge_weight_max`. Instead of taking the occurrences of both, we take the occurrences of the least occurring hashtag. For example, if #MAGA occurs 20 times and #taxreform occurs 100, and they occur together 18 times, their weighting would be:

$$= \frac{18(\text{occurences together})}{20(\text{weighting of MAGA, which has the least occurrences}).}$$
$$= \frac{18}{20}$$

This type of weighting would heavily favour niche hashtags, as the weighting is based on the count of the smaller hashtag. However, since we're filtering the mention count to be at least 200+ (discussed below), we'll find that for some specific hashtags, they weigh more accurately, but for others such as Trump, it results in a higher weighting value overall with each neighbour, which can be interpreted differently.

Finally, we have the `remove_min_count` function, which cleans up our graph and isolates for the most important/widely publicized issues. We found that in our cumulative 220 thousand tweets, we found an astonishing 38995 unique hashtags. We realized that across the political space, the wide variety of hashtags tweeted from personal matters, to misspellings is incredibly common. In order to better visualize our graph and increase its efficiency, the `remove_min_count` function is used to remove all the hashtags that aren't mentioned at least the min amount of times. This implementation was the source of quite a headache, required mutating a copied dictionary to not encounter mutation iteration errors, as well as removing all the neighbouring edges.

Obtaining Data-sets

The way to obtain our compressed file is to copy and paste the following link `https://utoronto-my.sharepoint.com/:f:/g/personal/jiajin_wu_mail_utoronto_ca/EsUilnfCdRdAsPfXj9nO_8sBuHV73C7rDo_R7GpVkgQHSA?e=KlGQrf` (It is a link for onedrive, the file in there is about 6GB). When trying to download, please click data.zip. In there you should be able to see a download button and download it.
Click what is in the PDF, or copy and paste that.
There are two zip files in the OneDrive. The reason we did that is because the hydrated tweet file is over 30 GB, and the TA might not have enough time to download it. When you finished downloading it, please extract it to our project folder to use it. In suggest data file, total_filtered_politician.csv is the processed csv file we used for our programs, and this is the only file we used in main. All the other files are used for making_new_csv.py to process the original data. Both full_member_info.csv and accounts-twitter-data.csv serves as sources to find us-politicians' information. Test_tweets_ids.jsonl only has around 2000 tweets. This file is suggested to TAs for testing purpose. The there is another zip file called all_tweet_ids.zip. This file is the raw data obtained. It is for TAs if they would like to see the whole process and doesn't find spending maybe an hour on the process. All_tweets_ids.jsonl is the fully hydrated tweets file with over 10 million lines of tweets. This file is not suggested for TAs to do testing on. With a older computer, the run time may take up to half an hour.

Visuals/Additional Libraries

Our visuals focus on allowing users to interact with our graph, and view it in different ways. We render all the vertices and their connecting edges, deciding the size and colour of the vertices based on the number of times the corresponding hashtag is used, and the partisan bias rating of the hashtag, respectively. We also provide the user the ability to select a hashtag from a dropdown menu implemented using Tkinter's `Combobox` class, and we render it and it's immediate neighbours in the browser (Amos). The user can also limit the number of vertices shown, which we implement using Tkinter's `Entry` class (Amos). Finally, we allow the user to choose to render vertices in the browser, which are randomly chosen, with preference towards showing connected rather than non-connected vertices.

In order to render our graph, we first convert it into a NetworkX graph, and then compute the positions of each vertex. We use NetworkX's spring layout, which means vertices are closer together when their edges have higher weights (Networkx.drawing.layout.spring_layout). We then create two scatter plots, using Plotly's `Scatter` class, with one representing our edges, and the other representing the vertices (Plotly.graph_objects.Scatter). We based the code for visualization off the code for rendering the graph from Assignment 3. We also use Plotly to provide the user a vertex's corresponding hashtag, how many times it was used, and it's partisan bias when the user hovers over the vertex.

$$Libraries : tkinter(\text{Included in Standard Python Library}), Networkx, Plotly \tag{1}$$

‘

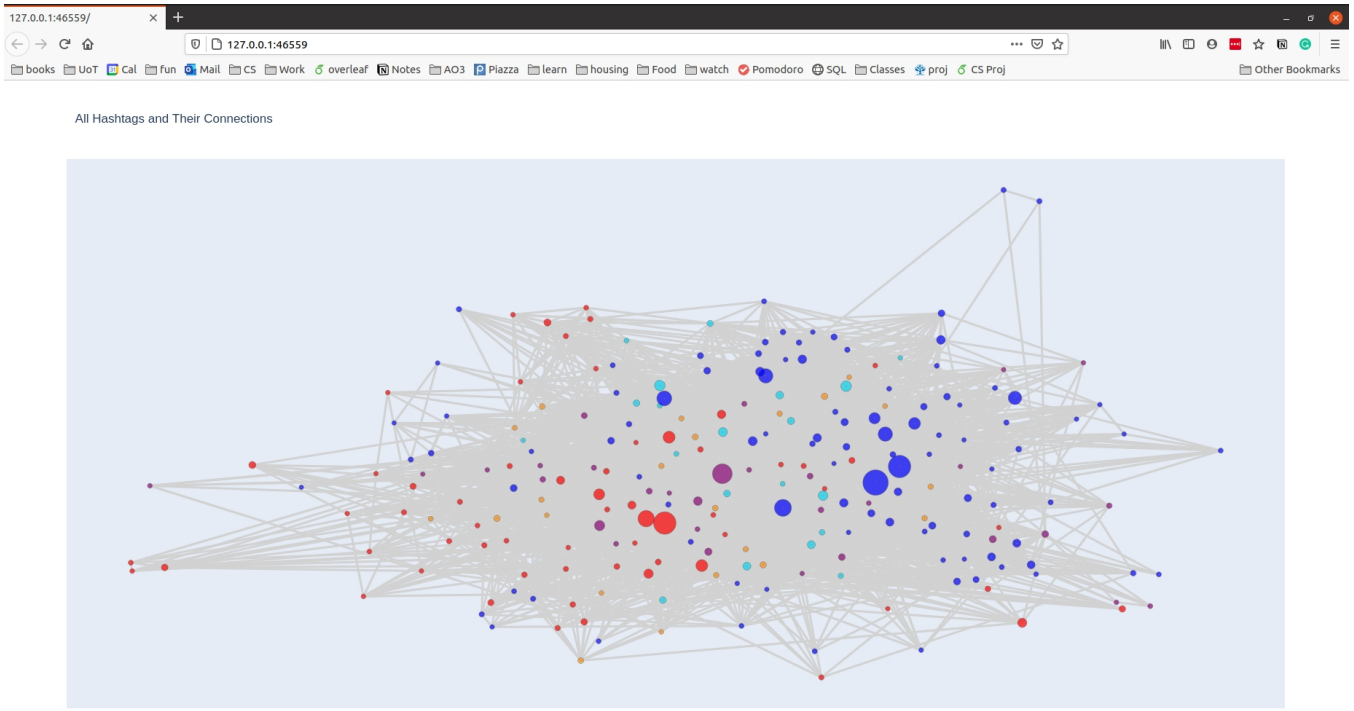
We will use *networkx* because our project is built around representing hashtags and their connections, which means we should represent our data in a graph. Since our graph doesn't need to represent "direction", we will convert our graph into the `Graph` class, which is for undirected graphs(Tutorial). We also take advantage of NetworkX's ability to store properties in a vertex, to store a vertex's partisan bias and the number of times it has been used. We also use NetworkX's ability to store weights for edges.

Instructions for Running the Program

First, ensure that you have followed the instructions from the section "Obtaining Datasets", and that the files from that folder are in the same directory as the file **main.py**. Then, install the required libraries specified in **requirements.txt**.

The file **main.py** is split into 3 parts. The first part is for creating the filtered csv file to pass to our graph. Since it takes over 20 minutes to process our entire 40GB file of hydrated tweets, we have put the final filtered tweet csv in our first compressed zip, as a completed version of the data generation that you can use to generate the graph without having the generate the data. This means that the first part is optional, but further instructions are in the file.

Afterwards when you are running **main.py**, after selecting a weighting model for the edges (more information in the comments of the file), it will open two things. The first, which will open in your browser, is the main graph of all the vertices.

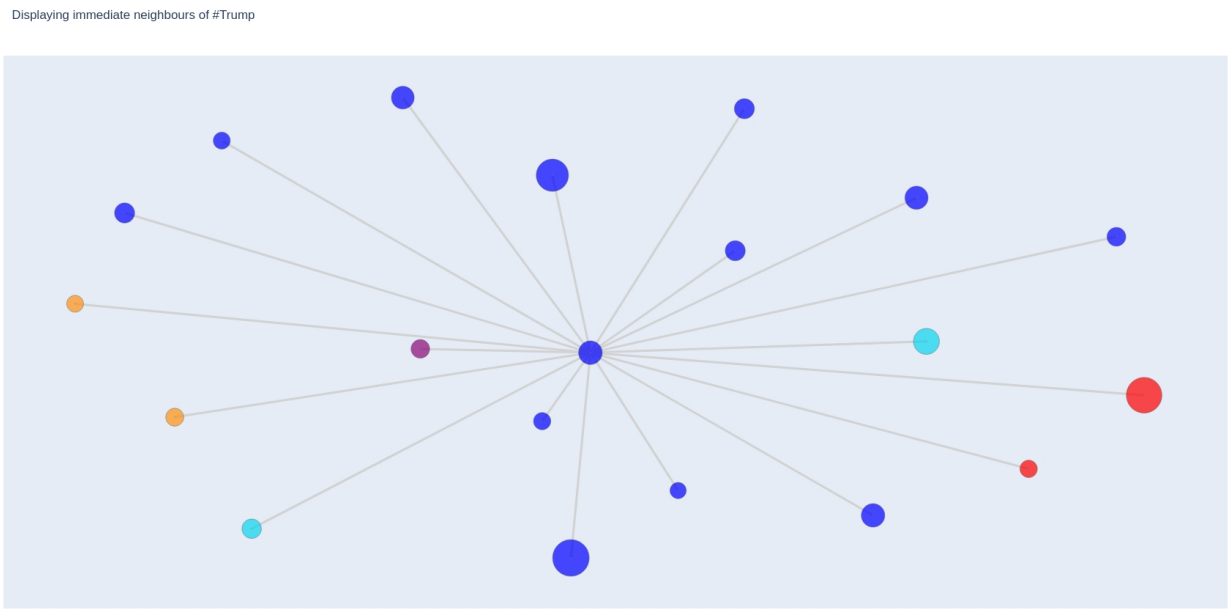


This shows all the hashtag nodes and their edges. If an edge between two hashtags is highly weighted, they are rendered closer together as they are more similar in topic. See if you can find #DACA cluster, or the #EqualPay cluster. Hashtags with a higher weight (meaning they occur more frequently) are rendered larger in size. Blue vertices are extremely democratic, light blue vertices are moderately democratic, purple vertices are moderate, orange vertices are moderately republican and red vertices are extremely republican. If you hover over a vertex, you can get more information such as how many times it has appeared in a tweet.

The second window is the interactive GUI that allows you to pull up specific hashtags its neighbours.

You can specify the number of nodes to display by typing in the box next to "Number of vertices to display". We recommend leaving it at 20. Above it, the "Show neighbours for" shows a drop-down menu that can be used to select a specific hashtag. Upon selecting, it will render a graph in a new window showing the vertex corresponding to the hashtag you chose, it's neighbours, and the edges to it's neighbours in the browser. At most, it will show the number of vertices specified in the box next to "Number of vertices to display".

Here is an example of the graph rendered for the hashtag 'Trump':



The button "View Random vertices" will render a graph with the number of vertices specified in the box next to "Number of vertices to display", as well as the edges between them. This graph will open in the browser.

Feel free to explore different hashtags, and see what popular topics may come up. Also, see what differences the two weighting systems have on the final graph.

For example, in the absolute weighting, #BorderSecurity an uncommon but extremely Republican hashtag has a relatively low weighting with #DACA, a very common mostly Democratic hashtag. However, when you switch to the 'max' weighting, the edge weighting has a very high value due to it prioritizing the niche, but usually mutual occurrence between the two.

Other topics you might find interesting are #China, #metoo, or #Parkland. When you're done exploring a hashtag, you can simply exit the window, and bring up a new one using the GUI! You can also open up multiple and check any comparisons between them.

Changes from Proposal

From our proposal, we had some important details to revise, which mostly involved being more clear about the data structures of our project, as well as where we could advance in complexity.

The first part was addressing a comment about how we're going to display our graph, which we did not really address in our proposal. With graphs specifically, you really have to strike a balance between a graph that shows the complexity of the data, while still be clear enough for a user to use. I think our final visuals excelled with our easy-to-use UI, as well as the ability to look specifically from node to node. Initially, we had a lot of issues with NetworkX attempting to only a few specific nodes, but we ended up coming up with a creative solution utilizing a GUI.

Our second concern was clarifying what the nodes and edges actually represent. I think we made it very clear that the hashtags were the nodes, with the weighting for each node being the overall "bias" towards a specific party. We also clarified that the edge weights measure the similarity of two hashtags. Leading to our last area where we improved.

Our last comment with David involved the complexity of the overall project, and how we would weigh the edges themselves. We looked into multiple options, and eventually settled on the final two that showed how there may be multiple good ways to weigh edges depending on your use. I think our implementation of the algorithms fit quite nicely as we're able to very easily "hot-swap" from the main class. It also brings more dynamic into what the user considers to be of most importance.

Discussion

Answering Research Questions:
We did a very successful job in finding and also representing how biased a hashtag may be towards a specific party. This is represented through our data-classes, as well as visualized through our graph in colours. A blue dot would mean that hashtag is most commonly in Democrat discourse, and red dot means its more of a Republican related issue. Any color that is in between means they are not as extreme as the previous ones, but orange would be leaning more towards republicans than light blue for example. In addition, hashtags that have a higher similarity weighting to each other are closer in proximity, utilizing the intuitions of spatial displays as opposed to having edges be another number on the screen. The size difference for the dots are also affected by how many times they show up as a hashtag. The more times they show up, the bigger they are.

Limitations:
One of the challenge we encountered while processing the raw data is that the raw data-set is so large. and included different countries, so it took a long time to get the data we have processed. Our original implementation took over 8 hours to process a million tweets, but eventually we found that was due to memory errors. By streamlining our pre-processing so that would would read a line and write a line, we cut the time down to approximately 10 minutes.

Another difficulty we encountered is the original `full_member_info.csv` implementation only provided members of the House of Representatives. This meant that instead of the 220 000 tweets we have now, we only had around 50 000. While this didn't actually impact our results, as we were looking for a holistic view of the political Twitter landscape as opposed to one specific person, it meant that we had less data to work with. Another similar problem was that we found out that politician's twitter name may not be the same as their real name. We had been using their real name to filter politicians and missed out on key political voices who had Leader or Senator in their Twitter handle. Luckily, we found that the politicians' `twitter_ids` are also available on both `full_member_info.csv` and `accounts-twitter-data.csv`. This means we could filter them by their twitter ids which won't be different.

We also had trouble with running our visualization algorithms fast enough that using our GUI felt interactive. Our solution was more intelligently designed algorithms, as well as removing vertices that don't appear more often than a certain threshold. We originally rendering graphs in a Tkinter window, using Matplotlib. Although Matplotlib is compatible with Tkinter, we faced various graphical issues, like vertices not fully rendering on the Tkinter window. This led us to switch to Plotly because Plotly supports pop-up labels for vertices from hovering over them. This was important because we wanted to be able to show graphs with a large amount of vertices, and pop-up labels meant we did not need to waste valuable screen space on rendering permanent labels for vertices.

Our last major point is the bias that might be found from our data itself. Notably, our dataset is somewhat skewed towards the democratic party, with 138 thousand tweets collected from democratic sources, and 88 thousand tweets from Republican sources. While in the last 4 years, the 45th President Donald Trump is known to be a prolific tweeter, this disparity probably relies in the fact that the average age of the democratic voter has historically been lower than the average Republican voter(Hess). This means that Republicans devote more resources to traditional media outlets such as Fox News, while democrats are notorious for fighting their battles in the social media space.
Future:
Due to the modular nature of our project, for the future there could be a lot of different areas to increase the

complexity, from going beyond just US politicians and looking at other countries and their political systems, to seeing if we can measure not just bipartisanship. In addition, while we demonstrated multiple weighting systems for the edges and nodes, I would like to see if we could alter the weighting systems with a lot more depth, assigning specific values to each Twitter member based on their history/background. This would go beyond our initial question, and see if we could detect "far-left" or "far-right" hashtags that are usually correlated with more extreme viewpoints. Finally, I think we could improve on our visuals, adding more interactivity or utilizing natural-language-processing to allow a user to input a hashtag, and see where that might fit.

Conclusion:

Overall, this overall project really challenged each individual's computer science skills, whether that was an immense amount of troubleshooting through the data collection, finding algorithmic errors such as not removing the edges of removed nodes, or a visual implementation that balances functionality with the visual display.

We're really proud of our final project, which represents a large jump in complexity and understanding of computer science and its applications. A personal belief is that the best projects are the ones where the moment its completed, you spend a long time using it yourself. I have multiple tabs open that looked into the weighting of issues that I knew of such as DACA or climate change, or issues that I was not aware of the significance such as the Chibok Girls.

Overall, the entire journey from data-set processing, to graph implementation, to visuals really challenged our skills, and was both a stressful, but extremely fun way to end our first year of Computer Science!

References

Amos, David. "Python GUI Programming With Tkinter." Real Python, realpython.com/python-gui-tkinter/.

Barbera, Pablo. "Pablobarbera/twitter_ideology." GitHub, github.com/pablobarbera/twitter_ideology/blob/master/primary/dat twitter-data.csv.

Documenting the Now. (2020). Hydrator [Computer Software]. Retrieved from github.com/docnow/hydrator

Hess, Abigail Johnson. "The 2020 Election Shows Gen Z's Voting Power for Years to Come." CNBC, CNBC, 18 Nov. 2020, www.cnbc.com/2020/11/18/the-2020-election-shows-gen-zs-voting-power-for-years-to-come.html.

Jiang, Julie, et al. "Political Polarization Drives Online Conversations about COVID-19 in the United States." Human Behavior and Emerging Technologies, vol. 2, no. 3, 2020, pp. 200–211., doi:10.1002/hbe2.202.

Matuszewski, Paweł, and Gabriella Szabó. "Are Echo Chambers Based on Partisanship? Twitter and Political Polarity in Poland and Hungary." Social Media + Society, vol. 5, no. 2, 2019, p. 205630511983767., doi:10.1177/2056305119837671.

"Networkx.drawing.layout.spring_layout." Networkx.drawing.layout.spring_layout - NetworkX 2.5 Documentation, 22 Aug. 2020, networkx.org/documentation/stable//reference/generated/networkx.drawing.layout.spring_layout.html# networkx.drawing.layout.spring_layout.

Panda, Anmol, et al. "Topical Focus of Political Campaigns and Its Impact: Findings from Politicians' Hashtag Use during the 2019 Indian Elections." Proceedings of the ACM on Human-Computer Interaction, vol. 4, no. CSCW1, 2020, pp. 1–14., doi:10.1145/3392860.

"Plotly.graph_objects.Scatter." Plotly.graph_objects.Scatter - 4.14.3 Documentation, plotly.com/python-api-reference/ generated/plotly.graph_objects.Scatter.html

Summers, Ed. "DocNow/Hydrator." GitHub, github.com/DocNow/hydrator.

Tutorial. Tutorial-NetworkX 2.5 Documentation, 22 Aug. 2020, networkx.org/documentation/stable/tutorial.html.

University of Amsterdam. "What Can We Learn from How Parliamentarians Talk on Twitter?" Twitter Politicians - Home, 2019, twitterpoliticians.org/..