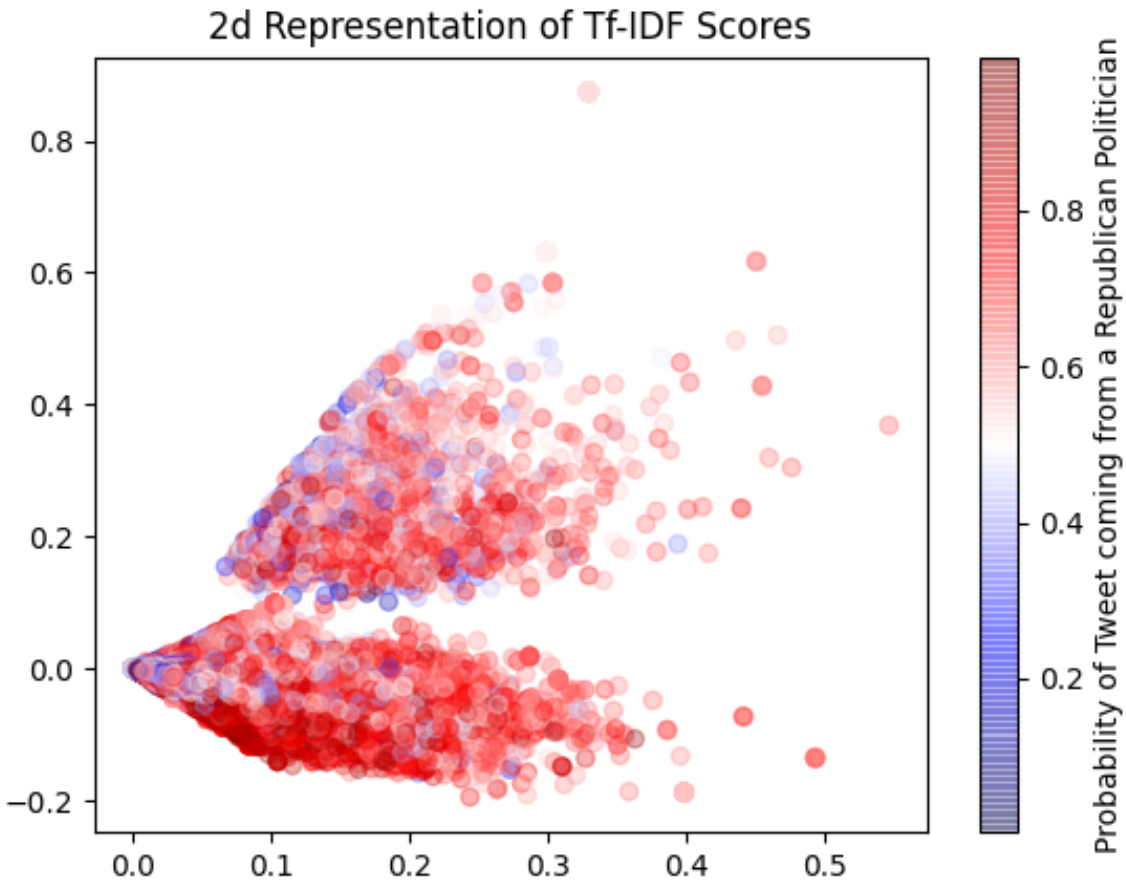
Political Tweet

Text Analysis Project

Liam Earley



Introduction

For this project, I analyzed a kaggle dataset of political tweets: the dataset comprised the 200 most recent tweets from official Democrat/Republican twitter accounts as of May 17, 2018. This dataset was ideal for several reasons: it was fairly small, it contained clear and balanced classes, and it was topical in the context of 2020. After processing the text, I trained a Naive Bayes classifier to distinguish between Democrats and Republicans based on Tf-IDF features. Using this model, I analyzed the most informative terms for each class and investigated individual accounts to understand how Democrats and Republicans tweeted differently.

Implementation

Early on in the project, I made several design decisions which had large impacts on the project. At the broadest scale, I decided to structure the program as a single script that would produce various reports and graphics for the reader. This came with several advantages: instead of designing a system of objects to interact with each other, I simply had to analyze the data and package up code into functions as I went along. After deciding to write a single script which generated an output, I then had to decide which functionality I wanted to implement myself and which I wanted to import from outside libraries. From prior experience, I knew that pandas would be great for data storage and manipulation, numpy is essential for anything involving numeric data, sklearn has great data processing and ML options, matplotlib can visualize data, and nltk covers most text processing not included in the other libraries. I opted to use all of these libraries except for pandas; pandas makes storing and manipulating data almost too easy for a good write-up, and I wanted to practice my python.

Since I couldn't use pandas for data storage, my next design challenge was to load and pre-process the data. The data was stored in csv format, so it made sense to load the file line-by-line, loading and pre-processing each input row one at a time. I wrote a series of functions: one to process tweet text, one to process a csv row, and one to load the csv file row-by-row and return the processed data. I wanted to be able to slice and iterate over my data easily (and transform it into numpy arrays for use with sklearn later on) so I stored the data in a list where each element of the list was a tuple. Storing the data in a tree/dictionary/etc. was possible, but a list excels for iterating and slicing. A tuple was preferable to a list to hold each row of data because a tuple is immutable; the elements of a list can be changed, while a tuple is set in stone. This protects against unintentional changes to the source data which could cause errors/affect the analysis. To make it easier to access the data stored in the master list, I also maintained 2 dictionaries mapping Parties/Reps (twitter handles, usually of

representatives) to their respective id numbers. Since these dictionaries were small, I implemented an anti_dict function to reverse them, allowing one to lookup handle/party in string format from an id number. I also maintained a third outside dictionary, Reps2Parties, which mapped handle ids to party affiliation ids for later analysis. I could've stored this in Reps (by making each value a list of [id, party_id]), but I wanted to keep Reps and Parties in the same format for abstraction purposes. Originally, I intended to maintain a Words dictionary of the form {word: # of times seen}, but sklearn's vectorizers rendered it superfluous.

After loading the data, I wrote functions to encode the text data (currently stored in tuples of words) as TfIDF/Count features using Sklearn's TfID and Count Vectorizers. These objects transformed each list of words into numeric numpy arrays for use with an ML model (turning sentences into vectors of word features based on the Vectorizer). I decided to use a Naive Bayes model as it is simple (and therefore works well with smaller datasets), adaptable, and traditionally works well with text classification problems. Naive Bayes models use word counts, Bayes' rule, and a "naive" assumption that features are independent ($P(X \mid C) = \prod_{i=1}^{n} P(Xi \mid Y)$) to predict the probability of a given sentence (as an n-dimensional vector X for vocabulary size n) belonging to class C. Since TF-IDF scores function as sort of an adjusted weighted count, and the conditional independence assumption already puts Naive Bayes on shaky theoretical grounding (P('Jude' | 'Don't', 'make', 'it', 'bad', Class X) != P('Jude' | Class X)), Tf-IDF scores can work well in practice even if they're theoretically somewhat unsound (much like Naive Bayes in general). After training the model, I used the probabilities P(Y | Xi) produced by the model to rank words in terms of importance, with more extreme probabilities (closer to 0 or 1 instead of 0.5) representing words more strongly associated with the closest class label (0 or 1). By generating a probability of belonging to one political party for each tweet, I was then also able to analyze trends in individual accounts/groups of accounts according to model predictions. In addition to this model, I also experimented with visualizing 2 and 3d projections of the TfIDF features, but did not find any meaningful clusters.

Results

The first step in analyzing the data was evaluating the classifier's performance on the data. The validity of any future analysis using the model is dependent on having a robust enough model to actually capture useful information…

```
Model Report:

              precision    recall  f1-score   support


           0       0.73      0.66      0.69      8506

           1       0.70      0.77      0.73      8786


    accuracy                           0.71     17292

   macro avg       0.72      0.71      0.71     17292

weighted avg       0.72      0.71      0.71     17292
```

As one can see from this report, the model achieved roughly 70% accuracy and fairly balanced precision/recall for the two classes. While the model yielded a 9% higher True Positive Rate for Republicans and a 3% higher Positive Predictive Value for Democrats, the model's F1 score and accuracy were convincing enough for me to continue ahead with this model. We don't need the world's most accurate classifier: we're looking for insights. Given that the model performed adequately, I proceeded to look at which words were most strongly associated with each class:

```
Top 10 Republican words:

0: #mn02

1: 50589

2: schumer

3: #gopfuture

4: #releasethememo
```

```
5: griffith

6: #politics

7: griffith's

8: lord

9: #millennials

10: sensenbrenner
```

The top 10 words associated with the republican party make sense circa 2018; in 2018, mn02 rep Jason Lewis lost to a democratic challenger in a tight race. It's interesting that the losers of the race tweeted the most about it: a similar phenomenon could be seen in the Democratic words with "ne02," a race they lost closely in 2018. "Schumer" refers to Democratic Senate Minority Leader Chuck Schumer, while the hashtag "#releasethememo" recalls 2018's ongoing investigation into President Trump from a distinctly Republican viewpoint. "Griffith" refers to Morgan Griffith, VA senator who faced challenges in 2018... "lord" reflects the GOP's religious persuasions, and "#millenials" light-heartedly alludes to the divide between older and younger voters in the United States.

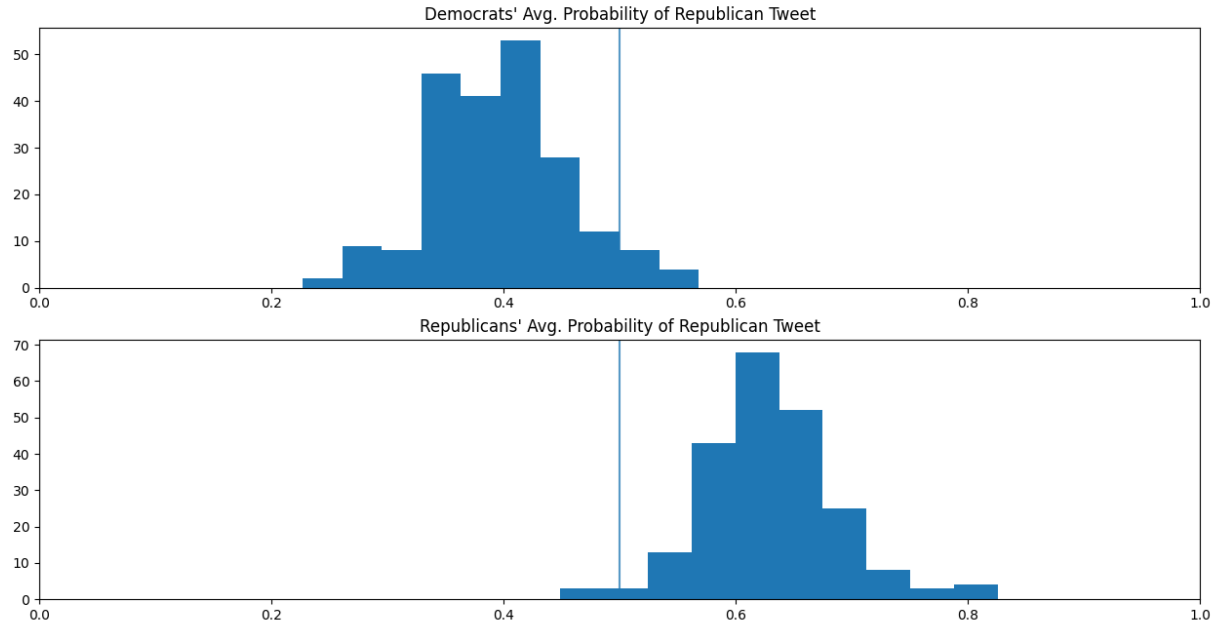
```
Top 10 Democrat words:

0: capuano

1: #protectourcare

2: #pollutingpruitt

3: equity

4: rent

5: #endgunviolence

6: #enoughisenough

7: #equalpay
```

```
8: #ne02

9: loebsack

10: #metoo
```

Democrat keywords offered similar insight: “Capuano” refers to longtime MA congressman Mike Capuano, who was defeated in a 2018 primary by Ayanna Pressley, the first black congresswoman from MA. “Loebsack” refers to Dave Loebsack, a fairly moderate Democrat who won his race in Iowa in 2018. Democrat hashtags mostly related to policy issues: “#protectourcare” refers to the ongoing political fight over the Affordable Care Act, “#pollutingpruitt” refers to EPA head Scott Pruitt who resigned in 2018 amidst growing bipartisan criticism for ethics violations. Most other hashtags are fairly self-explanatory. Other Democratic keywords included equity and rent, reflecting Progressives’ increasing influence on the political dialogue.

Comparing the two, we see some interesting similarities and differences. Both parties were more likely to tweet about their candidate when they won a contested race, but the district in question when they lost. This says something about the way politicians tweet: they steer the dialogue in the way that makes them/their party look the best. We see a similar effect when looking at the hashtags used by each party. Democrats used hashtags like #endgunviolence and #protectourcare to rally their base around key issues where they wanted to change the status quo, while Republicans used hashtags like #releasethememo and #gopfuture to promote the status quo’s benefits. This is consistent with the Democrats being out of power in 2018 and wishing to highlight things that need to change, and Republicans being in power in 2018 and highlighting successes/current issues that benefit the administration.

After verifying that the model worked and produced reasonable results, I continued my analysis by predicting every tweet’s probability of having come from a Republican account (No partisan bias here, the way my pre-processing functions worked just happened to assign Democrats label 0 and Republicans label 1). From there, I computed the average value of this probability for each Senator and plotted a Histogram of the probability for each party.




```
Party: Democrat Mean: 0.396 St. Deviation: 0.0582

Party: Republican Mean: 0.631 St. Deviation: 0.058
```

As expected, Republicans tweeted further to the right, and Democrats tweeted further to the left. Each party's mean probability was roughly 10% away from 50/50, with the Republicans only about 4% further away from 50/50 than Democrats. The standard deviation for each party was almost identical. The Republican distribution was more leptokurtic than Democrats', with more accounts concentrated around the mode and with longer, skinnier tails. This perhaps reflects how Republicans tend to organize around 1 ideology, with various individual politicians being further to the right or left of the platform. We don't see any evidence of a "pro/anti-Trump divide" in Repulican tweet patterns. The slight bump around the 80% mark for Republicans was caused by a handful of Congressional Committee accounts which inflated the count slightly at the high end. The Democrats' distribution has fatter tails, with more accounts concentrated around the 50/50 mark or the extremes than Republicans. Interestingly, the Democrats' histogram almost looks multimodal, with a dip in-between two peaks; this could perhaps reflect the divide between progressive and moderate Democrats, who likely tweet more

strongly Democratic/more Moderate respectively. It looks like the group of politicians most often misclassified by the model (on an averaged basis) is moderate Democrats: moderate Democrats and moderate Republicans tweet similarly according to the model. After analyzing each party, I decided to look at the accounts the model determined were the most Centrist/Republican/Democratic.

```
10 Most Republican Accounts According to Model

Twitter Handle, Average Probability of Tweet from rep being labelled
Republican

('FinancialCmte', 0.739144654749937)

('HouseSmallBiz', 0.7436373355381842)

('HASCRepublicans', 0.7489096861496034)

('WaysandMeansGOP', 0.7578615968133862)

('housebudgetGOP', 0.7657060680774239)

('RepMGriffith', 0.7862881506787565)

('HouseAppropsGOP', 0.7918550440185693)

('RepBuddyCarter', 0.794727246269865)

('RulesReps', 0.8027136897072681)

('HouseCommerce', 0.825736864891349)
```

The republican list was dominated by committee Twitter accounts belonging to Republican groups such as “HouseCommerce.” The model also picked up on Buddy Carter and Morgan Griffith, both running for reelection in 2018 in the swing states Virginia and Georgia. It makes sense that these two twitter accounts were tweeting strongly Republican during this time, as they needed to rally their base for the ongoing campaign.

```
Top 10 Least Republican Accounts According to Model

('RepCicilline', 0.22679874346707027)

('JuliaBrownley', 0.25696841802749354)

('RepDonBeyer', 0.26353683716334153)

('mikecapuano', 0.2843557398690399)

('RepRaulGrijalva', 0.2844419156726116)

('pedropierluisi', 0.28646928505595004)

('RepWilson', 0.28727625242467925)

('gracenapolitano', 0.28990757094721514)

('RepDanKildee', 0.2919390106812149)

('RepMaxineWaters', 0.29425591126136236)
```

The most Democratic accounts, according to the model, were generally more progressive Democrats such as David Cicilline (Vice-Chair of the Congressional Progressive caucus) or Don Beyer (Progressive Caucus member). We also see the reelection effect in action: Julia Brownley, a usually more moderate CA Democrat, tweeted much more like a Progressive during the 2018 reelection season.

```
Top 10 Most Centrist Accounts According to Model

('daveloebsack', 0.5008900687540396)

('RepSchneider', 0.5009836179013407)

('RepCurbelo', 0.5010391740545326)

('RepBrianHiggins', 0.5018132416998093)

('RepJimmyPanetta', 0.49690179716712307)

('RepStephMurphy', 0.5035809210035648)
```

```
('RepTomSuozzi', 0.5042326225918959)

('repdavidscott', 0.49402579959150955)

('RepDerekKilmer', 0.4904354785399381)

('RepPeteKing', 0.5111961211554676)
```

The centrist accounts are largely what you might expect given the distributions shown previously: mostly moderate Democrats like Dave Loebsack, a democrat from the conservative-leaning state of Iowa who is well-known for voting more conservatively than most Democrats. Most of these accounts come from very moderate/swing districts; Carlos Cubelo is the only Republican on the list of the 10, and his district swung Democratic in 2018. It makes sense that the centrist list is heavily tilted in favor of Democrats; from our model evaluation, we know that the model is more likely to predict Republican than Democrat. Since the model picked up on meaningful trends, however, we can still conclude happily having gained insight from keywords and histograms.

Reflection

Generally, I was moderately happy with my work. I felt that the code I wrote was good, but I struggled to find motivation or keep up with the timeline I set for myself. When I made progress, I was unable to keep up momentum like I usually am able to with coding projects/schoolwork. Putting personal issues aside, however, I feel that my project yielded some interesting results. While there were no world-shaking revelations contained within my analysis, I feel that the work I've done is a reasonable starting point for a more advanced project analyzing these tweets. While working, I frequently thought of ways I might extend the analysis if it were a personal project: potential avenues for future development include extracting sentiment from each tweet, clustering representatives, using a more sophisticated model than Naive Bayes, and using a more sophisticated feature extractor like BERT or GLOVE embeddings.

The actual development process, when unobstructed by my own shortcomings, went smoothly. Since I was developing one function at a time and packaging functionality up into the functions, I was able to unit test on the fly by testing a function with a variety of inputs before moving on to the next function. In one function I even left in some test code as a "sanity check" to demonstrate the model's functionality and make sure its output makes some degree of sense. As I developed, I realized that it was difficult to think of every edge case and problematic input that could arise when working with the data. For instance, early on I was plagued by encoding errors arising from

emojis in tweets that I never had to deal with when parsing homemade test strings. Furthermore, since I'm used to developing in Jupyter Notebooks, I originally planned to "test-as-you-go" the whole time and quickly ran into annoying load/preprocessing times. As such, I wrote a function to save the current data and model to a pickle file, which could then be loaded in if necessary to test new functionality without waiting for the whole dataset to load and be processed. While this patchwork approach worked, and has worked for me in the past, I now realize how clunky it really is. In the future, I'll consider writing dedicated test functions to speed up my development workflow. Overall, however, I feel that my development style was solid, and I'm excited to take on the next challenge.