CS182 Final Project: Analyzing Partisanship in Tweets Using Classification, Clustering, and Text Generation

Jessica Edwards, London Lowmanstone, Kit (Hong-Long Nguyen)

January 24, 2019

1 Introduction

From the game of Go to operating motor vehicles, computers are slowly yet maturely stacking up the list of tasks at which they are better than humans. But, for every methodical field machines can conquer, there is an intricate domain that seems only possible with a human touch, like poetry, or humor.

Intrigued by this trend, our project seeks to find out whether political partisan bias can be categorized and/or reproduced by existing AI methods. To conduct this investigation, we turned to natural language processing's favorite platform - Twitter. Using tweets as our training data allows for ample of reasonably sized, yet meaningful data points.

We are specifically interested in seeing whether conventional classification and clustering approaches can accurately categorize political leanings of tweets, and whether text generation can mimic the language of politically motivated tweets of different parties.

For classification, we will see how well the Naive Bayes algorithm fares against tweets from Democrats and Republicans. In addition, we will use the doc2vec approach to vectorize these tweets and compare the classification performances of logistic regression, k nearest neighbors, random forests, support vector machine, as well as unsupervised k means and agglomerative clusters using the doc2vec model. For genereration, we will use neural networks.

2 Background and Related Work

2.1 Bayesian Partisanship Classification Approaches

One of the first steps of our project was to train a Naive Bayes (NB) classifier for text classification. Naive Bayes methods apply Bayes' theorem with the assumption of conditional independence between features given the value of a class variable. Different Naive Bayes classifiers differ due to the various assumptions they make about the distribution of the probability of a feature vector given the value of the class variable. Multinomial Naive Bayes and Gaussian Naive Bayes are the two classic Naive Bayes variants used for text classification. Since Multinomial NB is most commonly used in text classification analysis like this with concrete labels, we wanted to focus on this when comparing Bayes algorithms with other classification and clustering methods but also seeked to analyze other Naive Bayes methods as well. While Multinomial NB implements the algorithm with multinomial distributed data, Gaussian NB assumes that each value associated with each class has a Normal distribution. On the other hand, Bernoulli Naive Bayes requires features to be binary valued. This algorithm is helpful when using word occurrence vectors versus word count vectors.

Some methods considered while implementing these algorithms include defining functions to process tweet text and remove stop words in an attempt to improve accuracy and using a Pipeline class to manipulate data parameters and classification.

We developed this program using tutorials on scikit-learn on Multinomial, Gaussian, and Bernoulli algorithms as well as similar attempts of building Naive Bayes Tweet Classifiers [2] and sentiment analysis methods using Multinomial NB [9].

2.2 Comparing Non-Bayesian Partisanship Classification Approaches

For all non-Bayesian classifications and clustering algorithms, we have to represent verbal tweets into quantitative entities (vectors) in order to use conventional algorithmic approaches to classify and order them. While naive approaches like bag of words or term frequency-inverse document frequency can get the job done using simple word frequency, these approaches do not account for word ordering nor semantics, both of which are obviously critical in detecting political leaning. To remedy for this, we opt for the doc2word model.

The doc2word approach is quite a complex one, and the foundation behind its workings can be found in Quoc V. Le's and Tomas Mikolov's paper where the idea was first introduced as paragraph vectors [7]. In a nutshell, however, doc2vec iterates upon word2vec, an algorithm that transform words into semantically sensitive vectors that capture synonyms, antonyms, and even analogies (king to queen is like man to woman). Doc2vec pushes the word2vec idea even further, assigning a semantically sensitive vector to an entire meaningful string of words (individual sentences, paragraphs, or in our case, tweets). In non-political contexts, doc2vec has been found quite effective in detecting document topics and mood sentiments.

Besides Naive Bayes, logistic regression, k nearest neighbors, k means, and agglomerative clustering, all of which have been discussed in class, we also investigated the accuracy of random forests and support vector machine in classifying partisanship. The random forest method, in essence, randomly samples n subsets within the dataset and performs a best fit decision tree within each of those datatsets. Consequently, each decision tree, using its own data and feature set, votes independently on the classification of the data point, and a simple majority of trees will determine the output classification. One of the major benefits to using a random forest approach is it does not overfit since it equally weighs random samples with replacement within the dataset.

The support vector machine (SVM) approach is even more intuitive than random forests. What SVM does is simply build a hyper-plane splitting the data points in halves in such a way that maximizes the distances between the plane and the data points closest to it. When simplified down to two dimensions, imagine constructing a straight line that best separates two clusters of points apart.

2.3 LSTM Text Generation

Long short-term memory (LSTM) units are the current method for neural network text generation. Proposed by Sepp Hochreiter and Jrgen Schmidhuber in 1997, LSTMs are a step above previous recurrent neural networks, because they have an internal state that is passed to the network when generating each new output. [4] This internal state allows the network to "remember" inputs that occurred many time steps ago. This helps the network to correctly remember syntax, punctuation, and capitalization over long periods of time which is ideal for text generation.

A blog post titled "The Unreasonable Effectiveness of Recurrent Neural Networks" by Andrej Karpathy spurred the development of many LSTM text generation programs. [6] Our particular program, however, is based off of Cassandra Kane's Lyricized project. [5] Our system reads each character from the input file and makes it lowercase. The vocab is formed of each of those characters. We use a 2 layer LSTM with 24 nodes in each hidden layer. The input and output layers are the size of the vocab. We trained for 20,000 iterations, with each iteration consisting of 64 batches, and each batch consisting of 100 sequential characters (also known as "time-steps"). The output is a random sample from the vocab based on the output probabilities of the network. We can run the network multiple times to generate a string of characters.

3 Problem Specification

This project aims to investigate how well conventional AI models and algorithms categorize political partisanship as well as whether text generation can mimic the language of politically motivated tweets of different parties.

Specifically, using the doc2vec model, can classifiers like logistic regression and k nearest neighbors accurately predict partisanship behind a politician's tweets? Do unsupervised clusters separate a body of tweets based on partisan lines at all? Does more training data improve the accuracies of said approaches?

With regards to LSTMs, can LSTMs accurately generate tweets that follow political leanings and correctly follow the grammar and structure of tweets? This will be successful if the output from the neural network looks realistic and it is not just copying its input.

4 Approach

4.1 Bayesian Partisanship Classification Approaches

After finding two different datasets (data.world's Politician Tweets [1] and Kaggle's Democrats vs. Republicans ExtractedTweets.csv [8]), we decided to split the 86.5k tweets in ExtractedTweets.csv categorized by party (half Democrat, half Republican) as both training and test data using scikit-learn's method train_test_split for testing Bayes algorithm methods. Splitting the tweet data into random test and train subsets, we set the parameter test_size to 0.33 so that 2/3rds of the dataset would be available to train the model and the remaining 1/3rd for testing the model.

Two different programs were used to test the classification accuracy when testing Bayes algorithms: tweet_classifier.py, multi_analysis.py, and bern_analysis.py.

In tweet_classifier.py, we have a class called TweetNBClassifier that has fit, predict and score functions for a Naive Bayes model. On the other hand, multi_analysis.py and bern_analysis.py use the Multinomial and Bernoulli methods, respectively, in the scikit-learn library.

tweet_classifier.py: This tweet classifier performs a binary classification where tweets are labeled as part of one of the two parties (Democrat and Republican). We then define functions ProTweets and rmStopWords to clean up the tweets and remove stop words from tweet text. After dividing up the ExtractedTweets.csv dataset into random test and training subsets, we define the TweetNBClassifier class to fit the classifier on our training data, predict the party (Democratic or Republican) of each tweet, and score the results by determining how many tweets have been classified correctly. Adapted from Kyle DeGrave's Naive Bayes Tweet Classifier. [2]

multi_analysis.py: After loading the ExtractedTweets.csv dataset and preprocessing the data, we can create a pipeline class to work with the vectorizer, transformer, and Naive Bayes classifier. Additionally, the algorithm tuned n-grams range, IDF usage, TF-IDF normalization type and Naive Bayes alpha during grid search which was launched with 10-fold cross validation. Adapted from Sergey Smetanin's Sentiment Analysis of Tweets using Multinomial Naive Bayes. [9]

bern_analysis.py: This file is very similar to multi_analysis.py except it uses
sklearn.naive_bayes.BernoulliNB instead of sklearn.naive_bayes.MultinomialNB.

Since Gaussian NB accepts dense matrices rather than sparse matrices which are not compatible with the CountVectorizer or TfidfTransformer methods, we decided not to change the algorithm significantly and to not test Gaussian NB. [10]

4.2 Comparing Non-Bayesian Partisanship Classification Approaches

As our raw dataset, I chose data.world's Politician Tweets (data.world/bkey/politician-tweets), which comprises of two csv files, one with over 1.6M tweets, each associated with its user's numerical id, and the other file with all these numerical id's which are then associated with particular accounts, position of power, and most importantly, partisanship. [1]

Realizing that there was no need to use all 1.6M tweets for training, I quickly put together a python script (csv_splitter.py) to divide the original tweets csv into datasets of 16,000 tweets each, labeled dataset1 through dataset16.csv. Datasets 8, 2, and 5 were randomly selected to become training sets, and dataset4.csv was the designated testing set.

From there, before all the tweets could be vectorized using doc2vec, they first have to be cleaned of punctuation, hashtags, emojis, hyperlinks, and more to optimize the semantic capabilities of doc2vec, as well labeled democratic and republican by cross referencing the csv containing all user id info. This was done using the csv2partisan_train8.py and csv2partisan_test4.py scripts.

After the aforementioned scripts loads the cleaned tweets into assorted txt files based on partisanship, training a doc2vec model using gensim's Doc2Vec module can begin. Check vectorize.py for detailed implementation.

Having built and saved the model locally, we can just load the model up to use for classification accuracy testing. I went with scikit-learn's API which provides a convenient means to run all the classification and clustering mechanisms. For each of the algorithms, I load up all the 100-dimensional training word vectors into a numpy array, and its corresponding labels into a separate, but identically indexed array. I then ran each of the supervised learning classifiers on the training arrays so they can best fit the training data. I then load up the test vectors and labels similarly into numpy arrays, and used the score() method of each classifier to determine how well each algorithm did categorizing the test data. To see these methods at work, see code in classify.py.

Lastly, for k-means and agglomerative clustering, I also used numpy arrays to feed the data into sciki-learn implementations of these unsupervised clustering algorithms. Since I opted for strictly unsupervised classification, as opposed to other semi-supervised approaches out there, I only fed the testing data into clustering to see whether clustering will naturally pick up on partisan division. The only challenge with this approach was evaluating the cluster's accuracy in partisan separation. The approach I independently came up with is one using the bitwise XOR operation. I labeled democratic tweets as "1" and republican as "0", and ordered both algorithms to only generate two clusters, also labeled "1" and "0". Using the labels_ attribute of the the algorithms, I could generate an ordered list of these labels, the elements of which I XORed with the elements of the actual correct labels to and summed them up to get the number of matches and mismatches, the larger of which is then divided by the total to get the final accuracy. This implementation, as well as the syntax and parameters for clustering, can be found in cluster.py.

4.3 LSTM Text Generation

For initial testing of LSTMs, a Kaggle dataset was used to train the algorithm on 300 tweets by Hillary Clinton around the time of the election. [3] These results were presented in our poster presentation.

For this report, the first 300 tweets in the Data.world "politician tweets" dataset that were explicitly said by someone who was a Republican or a Democrat were sorted into two files. Each tweet was separated by two newlines so that even if the LSTM failed to recognize that it should put two newlines between tweets, it would at least put one newline, enabling the recognition of the separation of tweets. The file that created the two text datasets is "politician_tweets_dataset_creator.py".

5 Experiments

5.1 Bayesian Partisanship Classification Approaches

We compared various Naive Bayes implementations: a Naive Bayes Tweet Classifier implemented without NB methods in scikit-learn which preprocessed the tweet data and removed stop words and a classifier that implemented Multinomial and Bernoulli NB methods as well as CountVectorizer, TfidfTransformer, and tuned parameters.

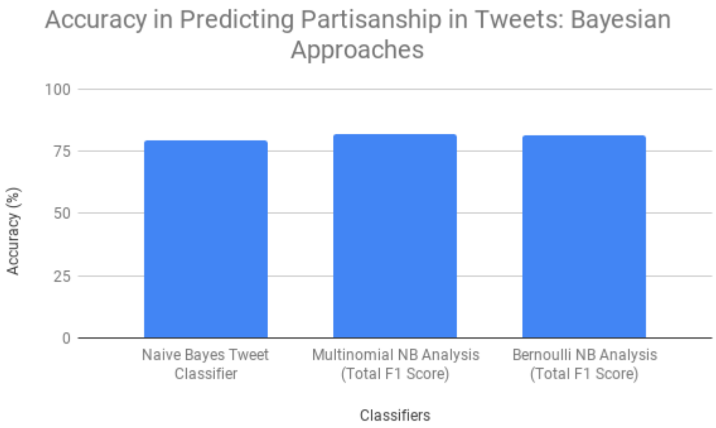
5.2 Comparing Non-Bayesian Partisanship Classification Approaches

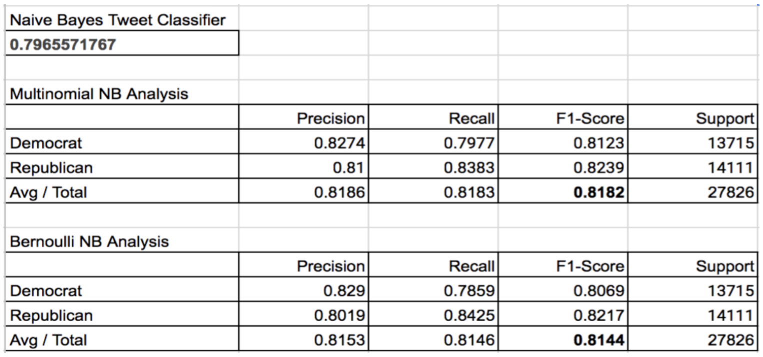
For this part of the experiment, we use the same doc2vec tweet vector model to compare the accuracies of each of the classification/clustering algorithms. We first used a small doc2vec model, vectorizing 30,000 distinct tweets, roughly half democratic and half republican, resulting in a corpus of 116,158 unique words. To observe the effect of size increase of the training dataset, we proceeded to train a large doc2vec model, this time vectorizing twice the number of distinct tweets (60,000) from both sides of the aisle, resulting in a corpus of 168,333 unique words. Each model was trained for 10 epochs to hopefully prevent both under- and overfitting. The testing dataset of 15,000 tweets were both times included as data points to train the model, although we have tested vectorizing completely new tweets (never seen by the training model) as well using the provided infervector() functionality provided by gensim's doc2vec implementation and found that this yielded quite similar accuracy results to having included the test dataset included in modeling.

5.3 LSTM Text Generation

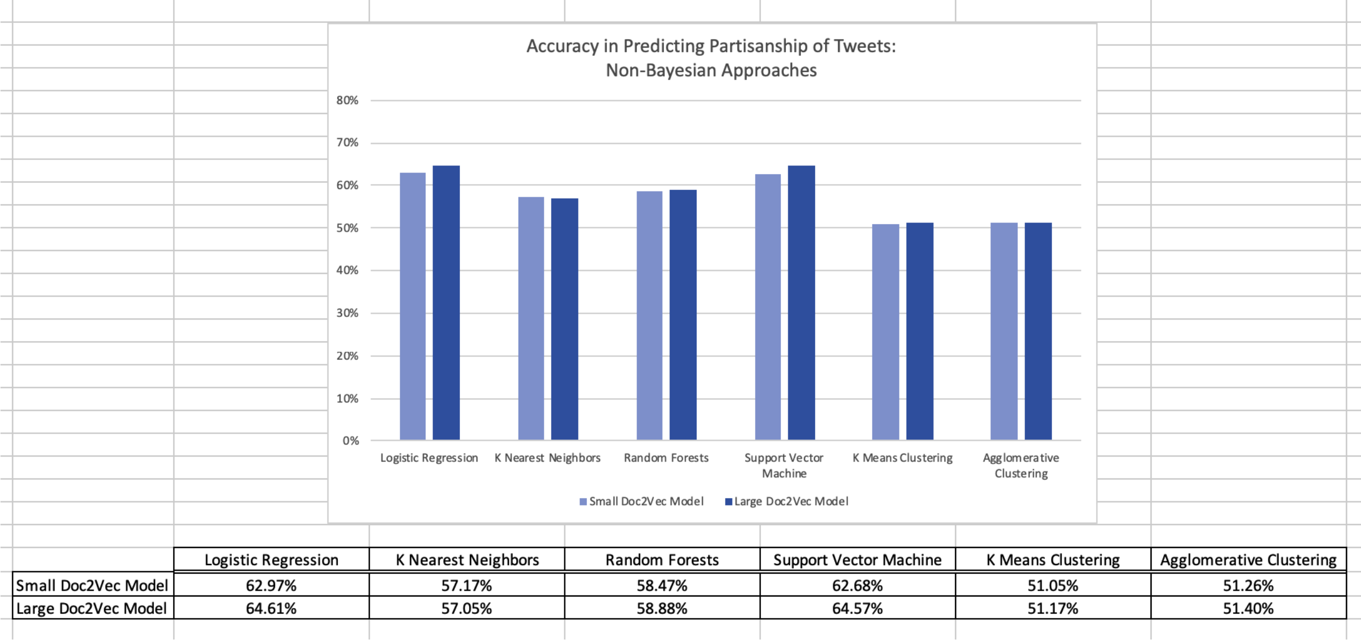
We used two text files, each containing 500 tweets, one with Republican tweets, and the other with Democratic tweets. As described earlier, two LSTMs were trained for 20,000 iterations on each text file. Afterwards, I generated 500 words from each LSTM, starting with a single space as the prefix, resulting in files containing tweets. These are in the folder labeled ”tweets” in the GitHub. The functionality to generate a certain number of words has since been removed and replaced with the ability to generate a particular number of tweets, each beginning with the same prefix, as this leads to more interesting data.

5.4 Results





Naive Bayes Tweet Classifier				
0.7965571767				
Multinomial NB Analysis				
	Precision	Recall	F1-Score	Support
Democrat	0.8274	0.7977	0.8123	13715
Republican	0.81	0.8383	0.8239	14111
Avg / Total	0.8186	0.8183	**0.8182**	27826
Bernoulli NB Analysis				
	Precision	Recall	F1-Score	Support
Democrat	0.829	0.7859	0.8069	13715
Republican	0.8019	0.8425	0.8217	14111
Avg / Total	0.8153	0.8146	**0.8144**	27826



	Logistic Regression	K Nearest Neighbors	Random Forests	Support Vector Machine	K Means Clustering	Agglomerative Clustering
Small Doc2Vec Model	62.97%	57.17%	58.47%	62.68%	51.05%	51.26%
Large Doc2Vec Model	64.61%	57.05%	58.88%	64.57%	51.17%	51.40%

6 Discussion

6.1 Bayesian Partisanship Classification Approaches

After comparing the Naive Bayes tweet classifier with Multinomial and Bernoulli Naive Bayes methods implemented using scikit-learn along with additional data transformation methods, the results show that all of the Naive Bayes methods tested classified data labeled as Democratic and Republican with about 80% accuracy overall as well as within each party.

There was a very small difference between the accuracy rates for all of the Naive Bayes methods, with the scikit-learn techniques being only about 2% more accurate than the Naive Bayes tweet classifier created without using scikit-learn.

In the future, rather than primarily focusing on the Naive Bayes method applied to the algorithm, we could adjust the parameters and determine whether adjusting parameters including n-grams range, IDF usage, TF-IDF normalization type and Naive Bayes alpha significantly changes the classification accuracy. We could also compare Gaussian Naive Bayes to the Multinomial and Bernoulli NB models and see if a Gaussian NB algorithm applied to partisan data significantly increases or decreases classification accuracy. Additionally, we could also test the Bayes algorithms on other datasets rather than splitting the large ExtractedTweets.csv dataset. While Bayesian algorithms seemed to have outperformed non-Bayesian approaches in terms of accuracy scores, with the highest accuracy percentage for a non-Bayesian algorithm being around 65%, it would be beneficial to see if using different datasets for comparing the two processes (Bayesian versus non-Bayesian approaches) played a significant role in creating the accuracy gap.

6.2 Comparing Non-Bayesian Partisanship Classification Approaches

Though all the experimental results for this section can be distilled into one compound bar graph, this by no means limit the number of insights we can infer from these results. First and foremost, let's discuss the general trend. Accuracies topping at 65% certainly suggests that the doc2vec vectorization does not separate tweets well into partisan aisles. None of the supervised classification algorithms seem to be able to do significantly better than random chance, though testing on multiple test datasets have suggested that these algorithms regularly perform about 10 percentage point better than sheer chance.

Regarding the individual classification approaches themselves, logistic regression and support vector machine seem to outperform the rest (around 65% accuracy), with both gaining slightly on accuracy with additional training data as well.

Unsupervised clustering approaches, on the other hand, did effectively no better than random chance. This can be easily explained by the following highly plausible hypothesis: as previous work with doc2vec outside of the political twitter domain would suggest, doc2vec with its semantic approach tends to place objects of similar topic near each other. Since both democrats and republicans often debate over the same topic, only with a different sentiment (often differing each other in only one key word, for example do vs do not), it makes sense that when the tweets are split in two, roughly equal numbers of opinions lie of both sides.

The same hypothesis would also explain why k nearest neighbors (KNN) did the worst out of all classification approaches. We can imagine that a tweet on immigration for instance, would have as its nearest neighbors all other tweets related to immigration. By the methodology of KNN, it would just count how many of the neighboring immigration tweets are from Democrats and how many from Republicans to arrive at its classification. But we know that this is deeply flawed since that tweet's partisanship should only depend on what it expresses about immigration, and not at all on what other tweets on immigration are about. This would reflect why KNN's accuracy was at only 57%, barely better than random chance.

From the results, we can also observe that accuracies barely change with doubling the training set. We can then infer that a relatively small number of 30,000 tweets (or even less) suffices to build a robust doc2vec model for representing politician's tweets.

In closing, we conclude see that even one of the most state of the art natural language processing approaches, doc2vec, combined with supervised classifiers cannot seem to accurately predict the subtlety of a partisan bias behind a tweet.

With that conclusion, however, we raise several concerns and limitations that go with it. The first issue is the fairness of tweets within the training and test datasets themselves. By fairness, I mean the following: though these datasets contain only tweets from politicians, they also contain many non-political tweets as well. For instance, there are hundreds instances of "happy birthday" tweets in both the training and test datasets. The question is then, do we expect anyone, machine or human, to be able to deduce partisanship from "happy birthday"? The answer is obviously no, so all of these non-political tweets will cause an immense decrease in the observed accuracy vs the actual one we are concerned about. This is by far the largest source of error that I believe exists in the experiment.

Another inadequacy that makes us unable to generalize these findings lie in the fact that we only used one language processing/vectorization method and that is doc2vec. We assumed that doc2vec will yield the best results, but without explicitly testing other methods, like bag of words (BoW), term frequency-inverse document frequecy(TF-IDF), or word2vec without the add-ons of doc2vec, we cannot guarantee that one of these less complex results would not surprise us. After all, we could see from the result of the previous part that simple Naive Bayes seems to outperform all these language processing efforts.

Last, but not least, is an experimental design compromise in the interest of time to include testing tweets in training the doc2vec model. While this seems like a very common approach as far as the projects I have seen online, the scientist in me senses that this cannot be 100% ethical. In an ideal test setting, one should not expect to have seen the test data at all before asked to do the classification, though I do understand that once we receive the test data, it is fair that we can incorporate it to retrain the model, but still, we should, and can do better.

In the future, to improve the results, we should:

(a) Screen out non-political tweets from the dataset to have the observed accuracy better reflect the accuracy we care for

(b) Perform the same experiment using BoW, TF-IDF, and word2vec as well

(c) Use a test dataset never seen before by the model and use infervector() from gensim's Doc2Vec pretrained model to vectorize it (has been tested on small scale with quite good results)

6.3 LSTM Text Generation

The LSTMs trained for 20,000 iterations overfit for our purposes. If you examine the files in the "tweets" folder on the GitHub, you can see that these are exact copies of the tweets from the dataset (found in the "data" folder). If we would like to have LSTMs generate without explicitly imitating, is it important to use prefixes with words that are not in the dataset, and to sample from less likely characters more often. If we generate 20 tweets with the prefix "is", we can see that most of the tweets end up exactly mimicking the original data, while the ones that choose uncommon characters quickly veer off-course into non-English before returning back directly to the data. (These examples can be seen in the "extra data" folder on GitHub.) If we generate 20 tweets with the prefix "abc" which it is unlikely to have seen previously, it chooses phrases which don't represent English as often as the phrases following "is", since it is likely to have seen the word "is" in the data set.

Thus, LSTMs may overtrain, which can cause them to mimic the input exactly. However, as soon as LSTMs move away from the exact input, they no longer seem to be able to imitate English well.

It may be that a larger dataset would help with this issue, as it may encounter the same word used in multiple different contexts. Training for less time or not overfitting in this case did not seem like a reasonable option, as when the network did not perfectly mimic the training data, its output did not look like English - less training time would likely exacerbate that effect.

Overall, training on just 500 tweets is not enough training data for computers to generate their own politically-leaning tweets that look like English.

A System Description

A.1 Bayesian Partisanship Classification Approaches

Code and results for this section can be found at the following GitHub Repo: `https://github.com/jessie9111/CS182-final-project`.

Instructions are in the README.md file, but please feel free to contact me at jedwards@college.harvard.edu if any issues arise.

A.2 Comparing Non-Bayesian Partisanship Classification Approaches

Everything needed to replicate my results can be found on my GitHub repo at:
github.com/kitnhl/partisan-tweets-classification
For detailed instructions, please see the README.md of said repo. I double checked so everything should be working, but if a technical issue arises, please contact me at
honglongnguyen@college.harvard.edu so I can resolve it.

A.3 LSTM Text Generation

The code for this portion of the project can be found at:
`https://github.com/Jadiker/CS182-final-project`
Use Python to run the files named "main_rep.py" and "main_dem.py". This will display how many iterations are left in the training for the dataset on Republican tweets and Democrat tweets respectively. Training for 20,000 iterations (the default) on both datasets with only a CPU generally takes about 24 hours.

Once training is complete, you can generate tweets by running "main_use_trained" in order to create tweets and save them to a file. You can choose the start of the tweet and let the LSTM finish it. The "training_name" variable should be the same in both the training main program and the tweet generation main program. You can train and generate text on your own files by changing that variable and putting a text file into the "data" folder with that name and adding in ".txt".

B Group Makeup

Jessica Edwards - All work under the Bayesian Partisanship Classification Approaches section (analyzing Naive Bayes algorithms, classifying tweet data using Bayes methods, running scikit-learn Naive Bayes approaches, analyzing Naive Bayes classification results)

Kit (Hong-Long Nguyen) - All work under the Comparing Non-Bayesian Partisanship Classification Approaches sections, i.e. tweet cleaning and processing, doc2vec model building, running all scikit-learn's supervised and unsupervised approaches to categorize tweets between parties, and the subsequent graphing and analysis of classification accuracy results.

London Lowmanstone - All neural network and text generation work. Cleaned data into text files, generated new text based on being either Republican or Democrat by using an LSTM neural network.

References

[1] bkey. politician tweets. https://data.world/bkey/politician-tweets, 2017.

[2] Kyle DeGrave. A naive bayes tweet classifier. https://www.kaggle.com/degravek/a-naive-bayes-tweet-classifier, 2016.

[3] Ben Hamner. Hillary clinton and donald trump tweets. https://www.kaggle.com/benhamner/clinton-trump-tweets, 2016.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[5] Cassandra Kane. Lyricized. https://github.com/cassandrakane/Lyricized, 2017.

[6] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/, 2015.

[7] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[8] Kyle Pastor. Democrat vs. republican tweets. https://www.kaggle.com/kapastor/democratvsrepublicantweets, 2018.

[9] Sergey Smetanin. Sentiment analysis of tweets using multinomial naive bayes. https://towardsdatascience.com/sentiment-analysis-of-tweets-using-multinomial-naive-bayes-1009ed24276b, 2018.

[10] David Ziganto. Sparse matrices for efficient machine learning. https://dziganto.github.io/Sparse-Matrices-For-Efficient-Machine-Learning/, 2018.