IDS 703 Final Project: Classifying Political Speech

Authors: Erika Fox, Raza Lamb

Overview:

Our final project goal was to design a model to effectively identify the political leaning of natural language. We approached this as a text classification problem and obtained “tagged” data from Subreddits. Specifically, we extracted Reddit posts from three different Subreddits: r/democrats, r/Republican, and r/NeutralPolitics, and worked to build a program that could take a small body of text (i.e., a post), and identify whether it has a “Democratic,” “Republican,” or “neutral” leaning. This report describes two approaches we implemented and evaluated for this problem: a generative probabilistic model and a discriminative neural network.

Data Scraping and Cleaning

In order to extract Reddit data, we used the Python requests library to scrape every post and comment from the three subreddits of interest. The code for this portion of the project was, somewhat ironically, derived from a very helpful Reddit user’s comment from the r/pushshift Subreddit. Specifically, this code outputs a text file for comments and posts (separately), with each line as a JavaScript Object Notation (JSON) payload.

Given the scope of this project, we wanted to restrict the data we were using in several key dimensions. First, political text obviously changes over time, so there was an incentive to limit this analysis to a specific, and ideally recent time period. Based on this criteria, we limited data to posts within 2020. This time period seemed ideal for this analysis, given that the U.S. presidential election occurred in this year, as well as several significant historical events like policy shifts following the course of the COVID-19 pandemic. From here, we dealt with a few additional issues to avoid potential interpretive issues with the posts in our data. For instance, many posts on Reddit are simply memes or links to articles, with very little text, so we removed posts with a total length shorter than 100 characters or with links. Finally, we also recognized that a post on a subreddit is not necessarily the ideal way to label the data. For example, a person who identifies as a Democrat could easily post on the Republican subreddit. While this issue is not entirely resolvable, we operated on the assumption that most of the posters, commenters, and viewers on a given subreddit were in fact Republicans, Democrats, or neutral. Thus, the analysis was limited to posts with a non-negative score (more upvotes than downvotes) to try and limit the amount of inappropriately tagged posts.

After limiting the data in the above manner, there were only approximately 800 posts per subreddit available. To increase the available data, comments were also included. We limited comments in a similar fashion to posts, with one exception. When sampling the comments available, it was evident that comments were less moderated than posts, and in fact comments would frequently be from Redditors self-identifying as the opposite party. Thus, we slightly increased the score threshold for comments from 0 to 1 to be included in our data set. Then, we sampled 4000 of the eligible comments for each subreddit and added them to our dataset. In the end, there were 14,506 posts and comments in our data, with 4,937 from r/Democrats, 4,809 from r/Republican, and 4,760 from r/NeutralPolitics.

Generative Probabilistic (Language) Model

The first model we designed to solve the text classification problem is a Naive Bayes classifier. This model was implemented through scikit-learn. The model was initially tested on simple word counts and TF-IDF, and simple word counts performed better. Before computing term frequencies, accents were stripped, all text was made lowercase, and stop words were removed. The correct class was determined via multiplication of conditional probabilities.

Neural Network Architecture

The second solution designed to solve this problem was a neural network. The design of this neural network was drawn heavily from a Medium post written by Ruben Winastwan. The architecture of the neural network in this post was originally designed to solve a text classification problem with five potential classes: sports, politics, entertainment, business, and technology. Thus, a few key modifications were needed to fit this problem.

This model utilizes pre-trained Bidirectional Encoder Representations from Transformers (BERT), a state-of-the-art transformer architecture used for representing words. This model was implemented in PyTorch and the Huggingface transformers library. The specific architecture of this model is detailed below.

There are several versions of BERT available through Huggingface. In this case, we used the BERT base cased model, which uses 12 stacked layers of encoders and treats capitalized words and uncapitalized words differently. The first step of the model is to input a string, specifically a Reddit post, which is split into word tokens through the BERT Tokenizer. Then, the sequence of tokens is passed through the BERT architecture, and the output is an embedding vector of length 768 for each specific token. For this classification task, we utilize the first token of every sequence as the input for the next step. This token is identified as the [CLS] token, or classification token. This token is then passed through a layer with input size 768, output size 30, and activation function ReLU. Then this output is passed through another layer with input size 30, output size 3, and activation function softmax, which then represents a probability distribution over the three potential classes.

Synthetic Data Generation

Before applying both models to the real data scraped from Reddit, we wanted to evaluate their performance on synthetic data generated according to a probabilistic model. To generate the synthetic data, we utilized the two main assumptions of the Naive Bayes classifier. First, is the bag-of-word assumption, that the order of words contained in the document does not matter. Secondly, the Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. This greatly simplifies this language modeling problem and allows generation of data based on the conditional probability distribution of words within topics.

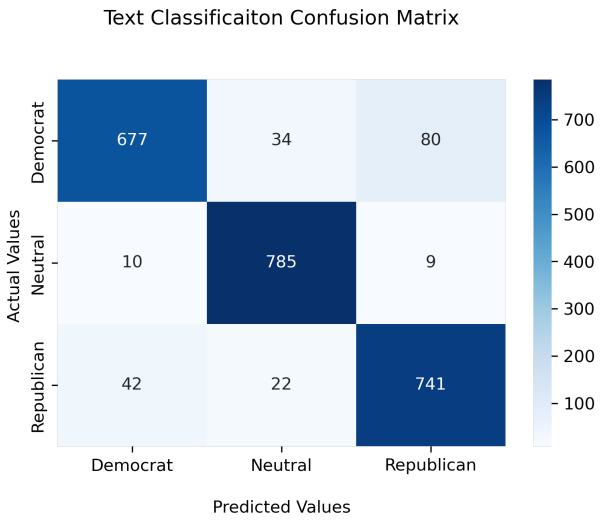
We utilized the CountVectorizer from scikit-learn to create a term-document matrix separately for each class of text, and then collapsed those matrices to get count frequency of words per

document class. These counts were translated into probabilities by dividing by the total number of words. Utilizing these conditional probabilities, we generated synthetic posts for each class, drawing the length from a poisson distribution with mean equal to the mean number of words contained in each class. While these posts follow the assumptions of the Naive Bayes classifier, they do not follow English grammar rules of logical structure. For reference, three generated posts are included below, one for each class:

- Democrat: “again polls the few either and kill countries primary but where trump the that tax before need just and is or mankind worst after with he not can schools lots like republican they so who assault better that not”
- Neutral: “session some guess showing it but gerrymandering president in horrifying variable point the be president testify just state the algorithms sure of they would hong actually and taxes states etcc worse mps maintain to explicitly since need acts number there spreading your the needs said waited tried have is without completely tens by you your being for bullet same government this 40 would governed the”
- Republican: “to have double than as been covid they from please to wasn us prompting old divide still they threats showing who saved read point look federal high the poor around 2020 for found are anti but don is be the was step organization”

Synthetic Data Results

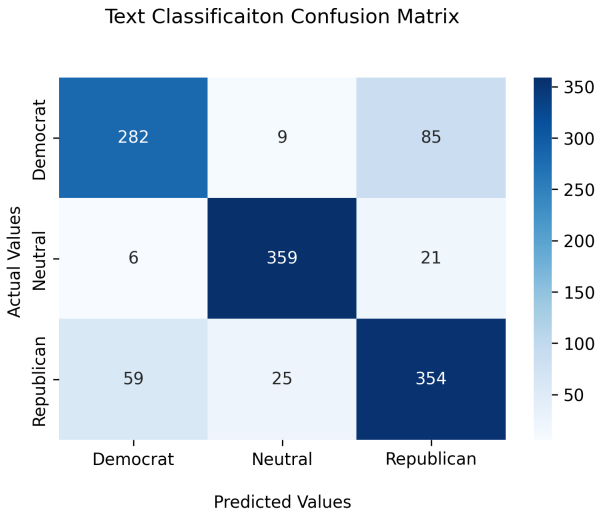
Once this generation process was complete, we trained and tested both the Bayes classifier and the neural network on our new synthetic data. We split our data for training and testing on an 80-20 split. After training, the Bayes classifier performed with high accuracy on our test set at approximately 92%. The confusion matrix is displayed below.



Interestingly, the classifier performs the best on neutral generated posts, and the worst on democratic posts. Also, interestingly, when the classifier makes mistakes, it is more likely to classify partisan posts as the opposite kind of partisan post, rather than partisan posts as neutral. We also calculated precision and recall for each category. Both democrat and neutral

categories had precision of 93%, while republican was 89%. Recall was worst for the democrat category with 86%, with republican better at 92% and neutral performing the best at 98%.

Next, we also trained the neural network on the synthetic data. Here, we split the data 60-20-20 for train, validation, and test. The validation data was used as the model was being trained to limit overfitting, and then the test data was used to evaluate. The neural net was trained over 15 Epochs, with batch size 4, learning rate of 1e-8, and weight decay of 1e-7. The accuracy for this method was worse, at approximately 83%. This confusion matrix is included below.

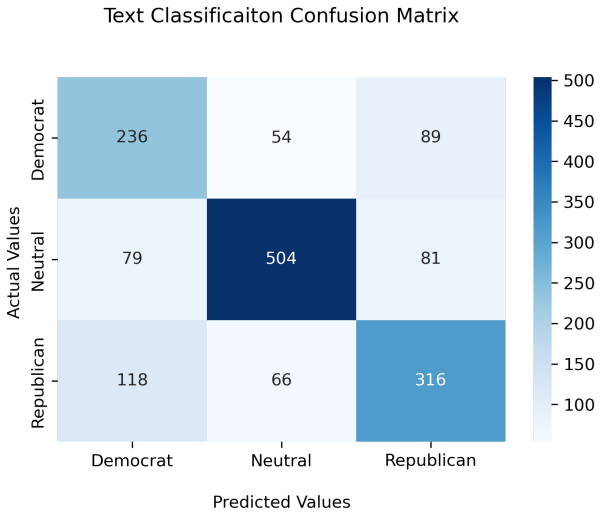


Visibly, there is a very similar trend to the Naive Bayes classifier. Clearly, the model predicts neutral posts with high recall, while again the model performs worst on democrats posts, and generally makes the largest number of mistakes when predicting partisan posts as the opposite party. Again, here, we extracted precision and recall. The model performed the best in both metrics on the neutral category, with 91% precision and 93% accuracy. The Democratic category had precision of 81% and recall of 75% and the respective numbers for the Republican classifier were 77% and 81%.

Reddit Data Results

After training and testing both models on synthetic data, we then trained and tested both models on the scraped and tagged Reddit posts. Not surprisingly, both models performed significantly worse on the real data. First, we tested and trained the Naive Bayes classifier on the Reddit data, using the same parameters as for the synthetic data. Here we obtained 68% accuracy, and the confusion matrix is also included below.

While the accuracy is overall worse, the patterns from the synthetic data continue, interestingly. Here, we noted that the precision and recall are the best for the neutral category, at 81% and 76%, respectively. The democrat class, on the other hand, has the worst precision and recall, at 55% and 62%. The republican class is not far off with precision of 65% and recall of 63%.

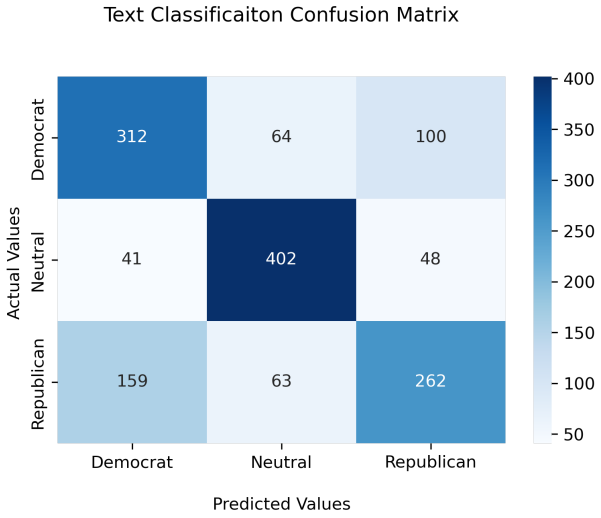


Due to the assumptions of the Bayesian classifier, we can interpret the results of the model by examining the words most likely to occur in each category. Below are listed the 10 highest frequency words by category (excluding stop words):

- Democrat: trump, people, like, would, get, biden, vote, think, one, right
- Neutral: would, people, like, gt, one, think, trump, could, even, also
- Republican: people, like, trump, would, think, one, get, even, know, right

The considerable overlap here sheds some light on why the classifier performs worse on the real data. Eight of the 10 most frequently occurring words are shared in the Republican and Democrat category, which could make classification very difficult, and suggests violation of the assumptions of Naive Bayes.

Following this, we also trained and tested our neural network on the Reddit data. The same hyperparameters were utilized in training, and the resulting accuracy was 67%.



Again, a similar trend is found, with the network performing the best at neural classification, with precision of 76% and recall of 82%. The democrat and republican categories were similar with 61% precision and 66% recall for democrat and 64% precision and 54% recall for republican.

Discussion

After evaluating the performance of both approaches, there were readily evident strengths and weaknesses. Overall, in this case, the Naive Bayes classifier seems to be a better choice for this specific problem, after considering several factors, including accuracy, computational requirements, and interpretability. Below the pros and cons of each approach are outlined in detail.

When considering the Naive Bayes classifier, it obviously performs incredibly well on the synthetic data. This is not surprising, given that the data is generated according to the assumptions of the classifier. The performance is much lower on the real data, which is not surprising. Especially with political speech, it's clear that the assumptions of the classifier are violated. For example, it's clear that the probability of the word "Trump," referring to the 45th president of the United States, appearing in a post of a specific category definitely depends on the other words in the post (i.e., positive vs. negative sentiment). Furthermore, the order of words also likely matters. This is the largest con of this approach for this problem. On the other hand, there are many positive aspects of this approach. Due to the simplifying assumptions, the interpretability of the classification is very clear. It's possible to easily extract the word frequencies by class, as was done to generate the synthetic data. Continually, this approach is computationally easy—training on the full dataset takes less than 10 seconds.

The neural network utilized in this analysis has very different pros and cons. A large potential positive of this approach is that it is not bound by any assumptions about natural language. Theoretically, this architecture could represent complicated dynamics, including word order, sentiment, and even potentially sarcasm (though not likely). However, in actual implementation, as seen above, the neural network did not perform better than the naive Bayesian classifier. Continually, the neural network has significant downsides: it is very computationally intensive. This specific neural network was trained on a GPU hosted on Google CoLab, and took approximately 10 minutes per epoch. Trained locally, the length of an epoch would have been closer to 24 hours. Additionally, the flipside of the benefit of no assumptions is a loss of all interpretability. The vectors representing each tokenized word have no inherent meaning, and cannot be interpreted.

There are a few important limitations to acknowledge in this analysis. The data utilized is not ideal. Reddit posts are likely not fully reflective of general political discourse, and as mentioned previously, there is no barrier to prevent a democrat from posting on the republican subreddit and vice-versa. Additionally, there are inherent challenges in classifying text posted on the internet. This speech is very likely to both contain spelling errors and contain abbreviations/slang. The intersection of these problems make spelling correction difficult without a greater undertaking, and likely influences the accuracy of both methods.

In theory, the best solution to this text classification problem inherently lies in the trade-off between the neural network and the Naive Bayes classifier. The neural network has the potential to have better accuracy, at the cost of computational intensity and loss of interpretability. However, in this specific implementation, the neural network does not display the capacity to gain a competitive advantage in accuracy, so the Bayes classifier is a better choice.