Classify Democratic / Republican Political Speeches

1. Problem Statement

What is the problem you want to solve?

I want to investigate models that can predict if a political speech is a certain affiliation (Republican or Democratic). I'll train different models that will be able to classify a given speech as Democratic or Republican *solely based on the text.*

Who is your client and why do they care about this problem? In other words, what will your client do or decide based on your analysis that they wouldn't have done otherwise?

This is a research problem. It doesn't have a direct business application. However, the model(s) that are the most efficient could be trained on different data so that they're able to classify other types of text. In other words, I'll be training the models on political speeches to predict affiliation, but they could be trained in any other version of labeled text data to predict other variables.

2. The Data

Description of the Dataset

The data needed for training and testing the classification models *are political speeches classified by affiliation (Republican/Democratic).* Once it's cleaned and wrangled, it'll have the form of a pandas dataframe with columns "Speech", "Label (affiliation)", and other metadata like "Author" or "Date", if available.

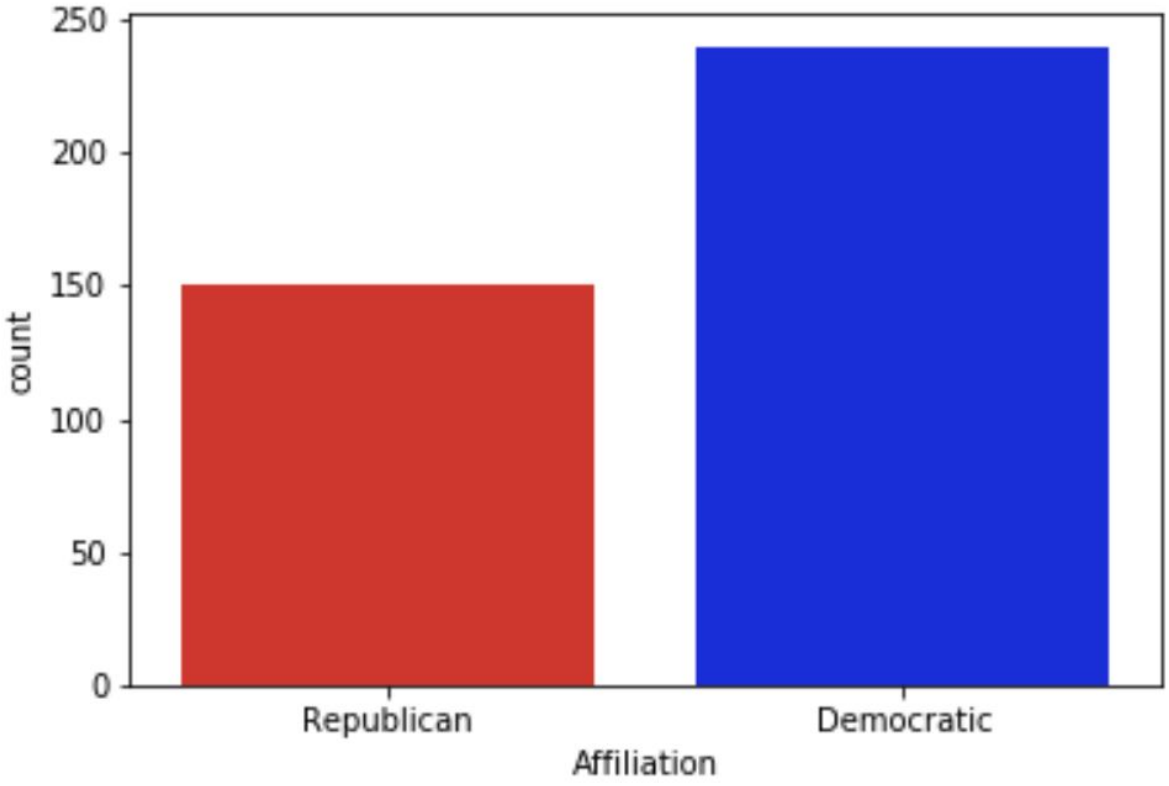
I've obtained the current data from the following website: millercenter.org/the-presidency/presidential-speeches. This website aggregates speeches given by American presidents over the years, from both the Republican and Democratic side.

The process of obtaining the data was as follows:

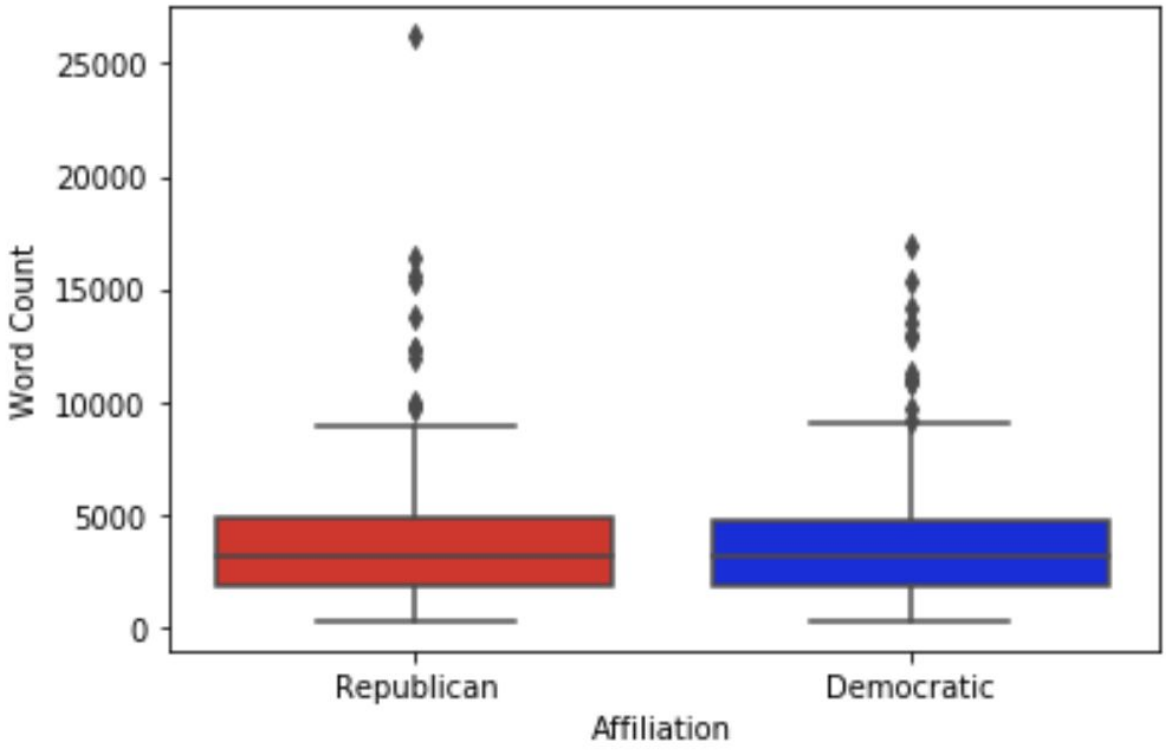
1. Identify all presidents I'm interested in getting speeches from. I only included presidents starting from the 50s, as anything before that might not be comparable.
2. The website uses JavaScript to show the speeches - you have to click a "View Transcript" button. I set up a script using Selenium that automates the necessary interaction with the browser to display the proper information (the speech) and then downloads it to disc.
3. Finally, once I had all the data in a dataframe, I proceeded to do the text pre-processing. In my case, it involved:
    a. Expand contractions ("don't" > "do not").
    b. Remove non-word characters
    c. Lemmatize the text using SpaCy
    d. Remove stop words using SpaCy
    e. Remove non-ascii characters

3. EDA

The following exploratory visualizations were performed to better understand the data.

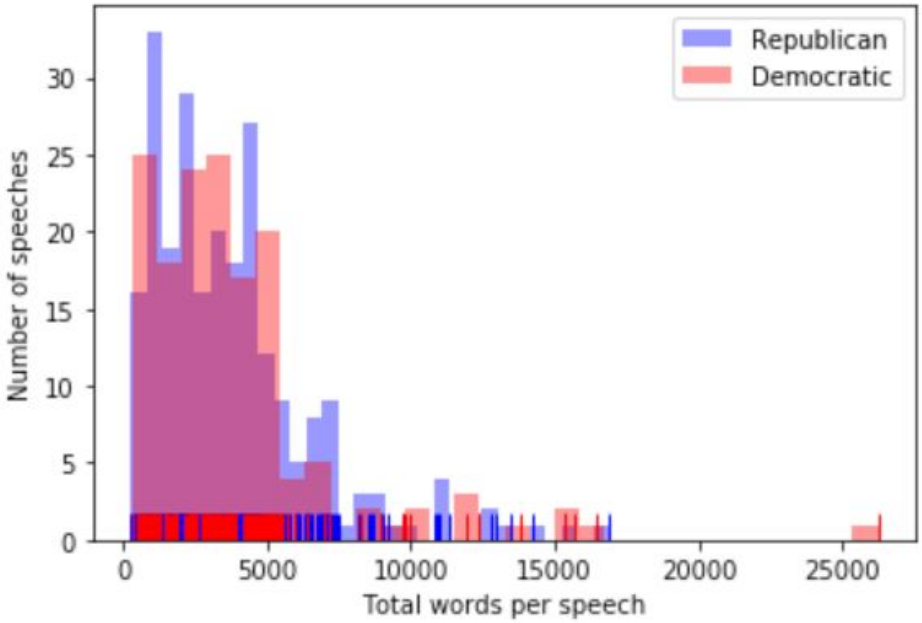


Count of speeches by Affiliation

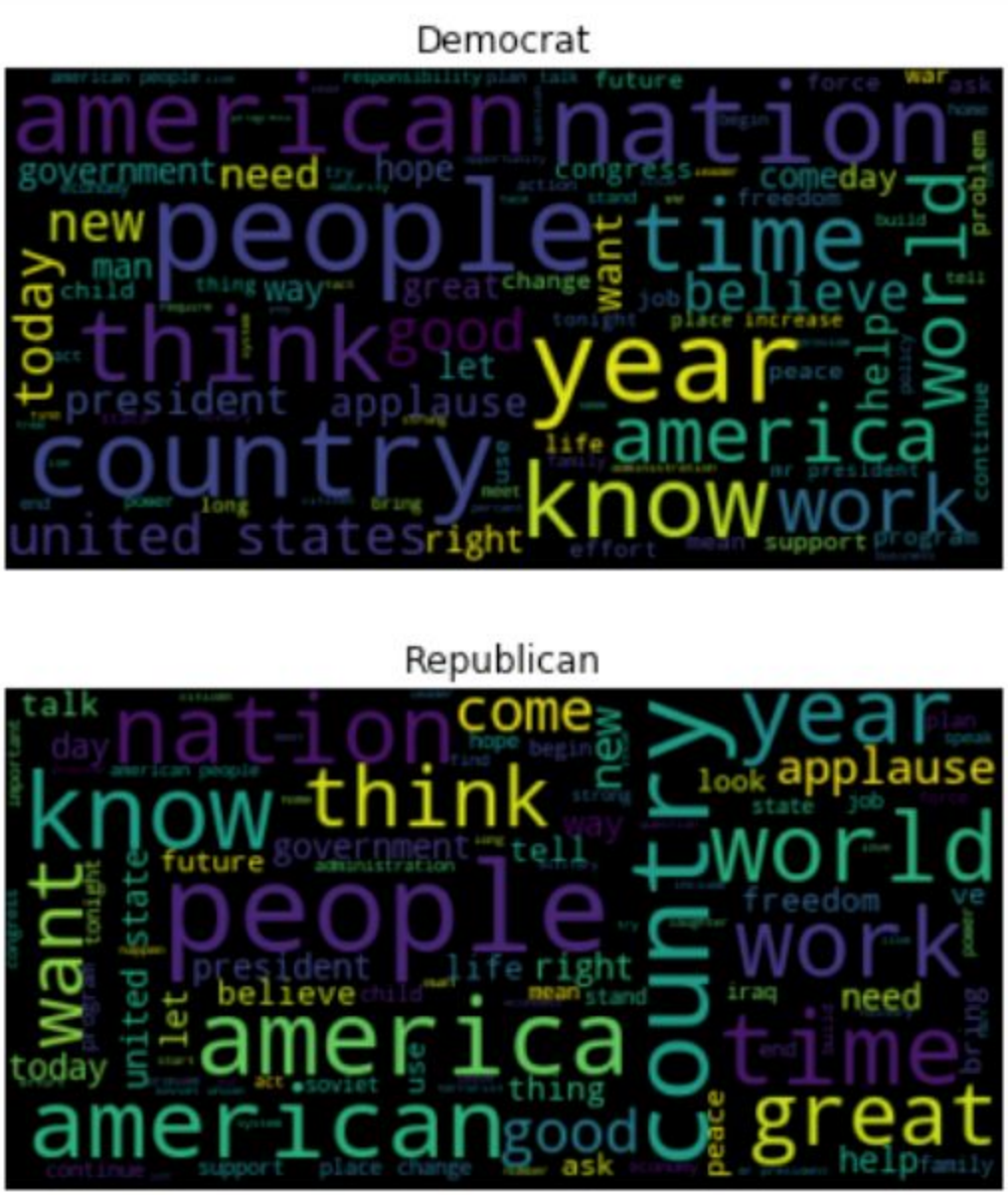


Distribution and summary statistics per Affiliation

Note that there's a big outlier on the Republican side - interestingly, it's a recent speech by President Trump on the state of events regarding the COVID-19 pandemic.
Otherwise, the distributions are very similar.



Histogram of number of words per speech



Democrat and Republican Word Clouds
of most common words

No further inferential statistics exploration was done, as we're dealing with text data, which has a very specific pipeline that involves *vectorization* and then *classification*.

4. Vector Representations of Text

The first step when doing Machine Learning on text data is to convert the text (words) to numbers (vectors).

There are two main approaches: “traditional” (count-based), and deep learning approaches.

The “Bag of words” model and the “TF-IDF” model (term frequency-inverse document frequency) are considered count-based approaches and were applied to this project.

“Bag of words” is the most simple vector space representation model for unstructured text. It represents each text document as a numeric vector where each dimension is a specific word from the corpus and the value could be its frequency in the document, occurrence (denoted by 1 or 0), or even weighted values. The model’s name is such because each document is represented literally as a bag of its own words, disregarding word order, sequences, and grammar.

The “TF-IDF” model tries to deal with some issues in the previous term-based approach, where there might be some terms that occur frequently across all documents and these may tend to overshadow other terms in the feature set. Especially words that don’t occur as frequently, but might be more interesting and effective as features to identify specific categories. TF-IDF is a combination of two metrics, term frequency (tf) and inverse document frequency (idf).

The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

Both models were applied using the scikit-learn library.

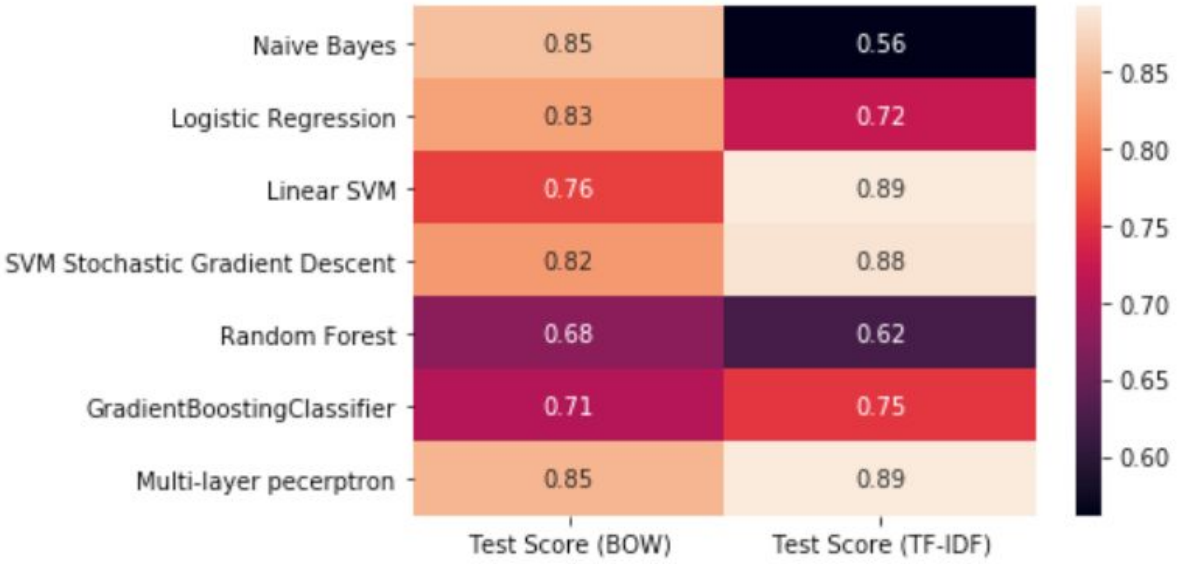
5. Machine Learning Classification Approaches

The following 6 standard Machine Learning classification algorithms were applied:

1. Naive Bayes
2. Logistic Regression
3. Linear Support Vector Machines
4. SVM Stochastic Gradient Descent
5. Random Forest
6. Gradient Boosted Machines

Additionally, a Multi-layer perceptron neural network was trained using the same vectors.

Here are the results of the 6 ML algorithms + MLP with each vectorization approach:



We can see that the best performing ML model was Linear SVM with TF-IDF vectorization, and the MLP neural network also had the same high accuracy score.

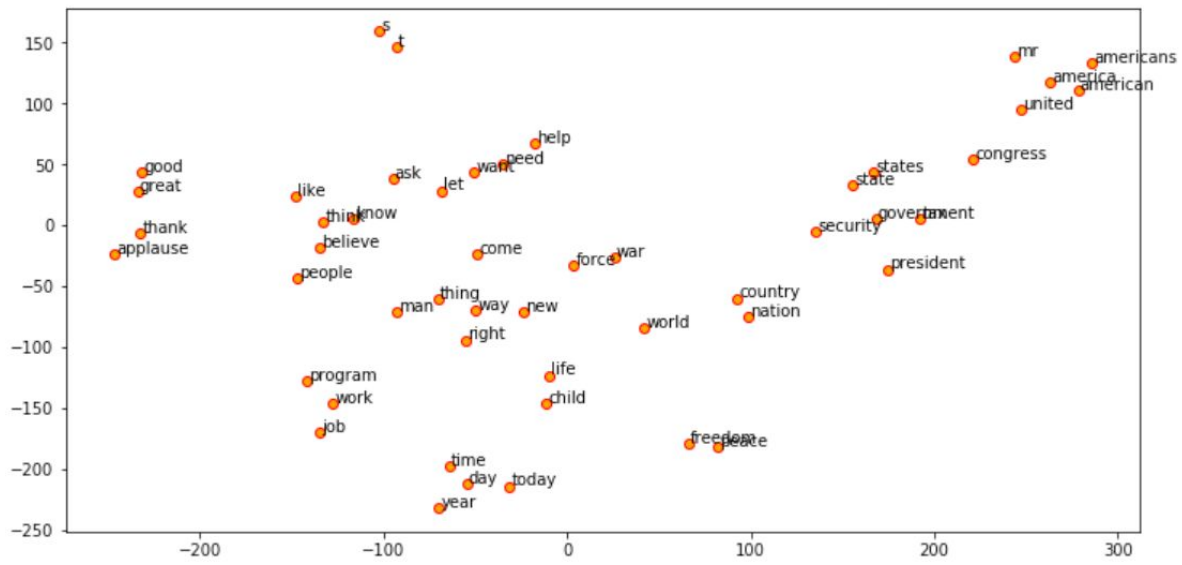
Parameter tuning through GridSearchCV was performed on that model, however the accuracy didn't increase and stayed at 89%.

6. Deep Learning Classification Approaches

The first approach I wanted to try was to use pre-trained vector representations achieved by using deep learning approaches similar to word2vec.

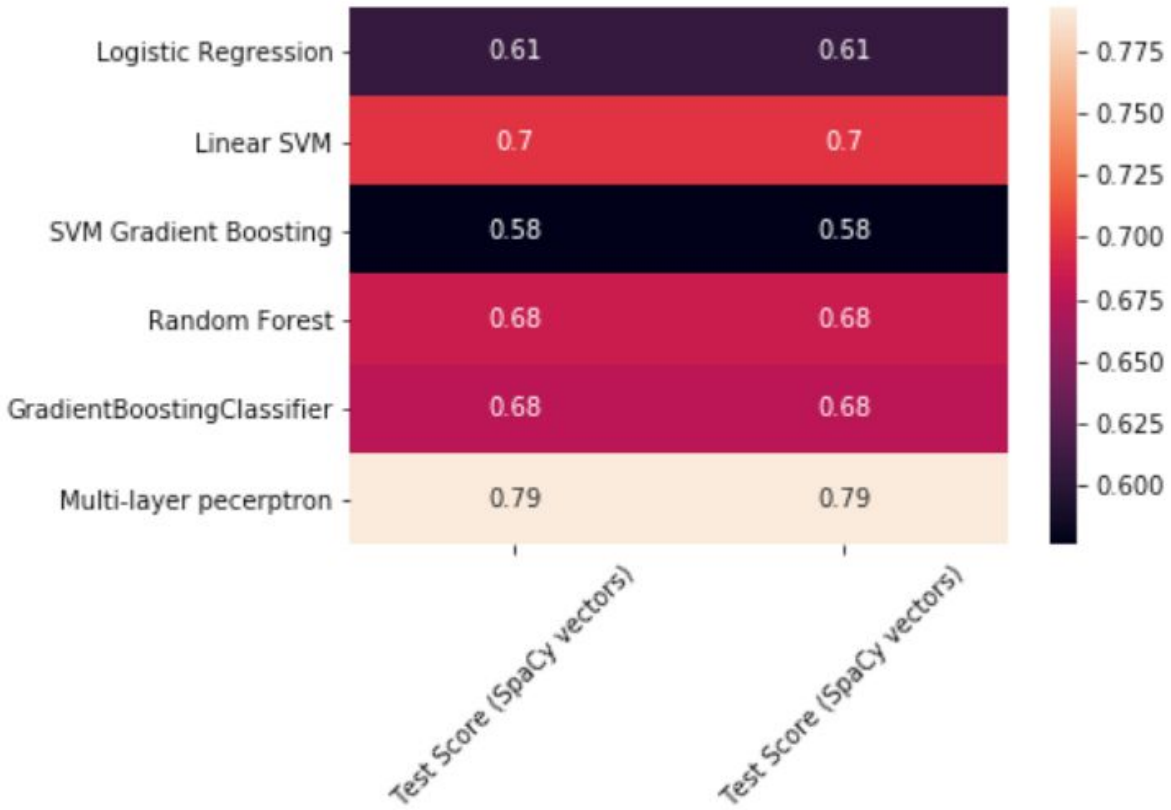
GloVe vectors + ML/MLP classification methods

I didn't use word2vec directly, instead I used the pre-trained vectors from SpaCy that were obtained by implementing the GloVe algorithms.



TSNE representation of the most common words in the corpus of political speeches. It's clear that similar words are found in an approximately similar vector space.

The same 7 classification methods were applied to the pre-trained vectors. Here are the results:



We can see they're not as good as with TF-IDF previously. My intuition is that it's because these are pre-trained vectors, and the speeches data is not too much yet, so the results are very general. Having more speeches as training data might help. Additionally, training our own vectors on the speech data will probably also increase the performance (once we have enough data to do so).

FastText vectors + CNN

Finally, a combination of pre-trained vectors from FastText (Facebook) and a Convolutional Neural Network was implemented.

However, the results were not very impressive. This is most likely due to the fact that, again, the amount of speeches we have is not too big, and deep learning approaches, in general, need much more data than traditional Machine Learning approaches.

In fact, a first approach with a *validation datase*t didn't achieve good performance, because it reduced the amount of training data even further.

Accuracy: 50.63%

	precision	recall	f1-score	support
0	0.00	0.00	0.00	39
1	0.51	1.00	0.67	40
accuracy			0.51	79
macro avg	0.25	0.50	0.34	79
weighted avg	0.26	0.51	0.34	79

Metrics of CNN with validation dataset

When the validation dataset was not used, the model overfitted, but it still achieved better performance overall.

Accuracy: 75.95%

	precision	recall	f1-score	support
0	0.95	0.54	0.69	39
1	0.68	0.97	0.80	40
accuracy			0.76	79
macro avg	0.82	0.76	0.75	79
weighted avg	0.82	0.76	0.75	79

Metrics of CNN without validation dataset

7. Next Steps

The following additions to the project can be attempted to see if they increase the accuracy and performance of the models.

1. **Ingest more data**. There are two more sites that I've identified with useful speeches: americanrhetoric.com, rev.com. The first one is especially appropriate, however, since the time this project started, it went down. A historic version can be found through the Internet Archives. This could be used to scrape more speech data.
2. **Train our own word vectors**. Again, this is somewhat dependent on having more data. But obtaining our own word vectors could potentially very much improve the accuracy of the end classification models.
3. **Fine-tune the neural networks' hyperparameters.** Both the MLP and CNN networks can be tweaked to try to improve their results.
4. **Host the best performing model as a web application and continue ingesting user-generated data**. By hosting the best performing model online and offering to the public, a reinforcing learning loop can be set up and the models trained with the new data users use as input to the web application.