**Capstone #2 - Introduction**

In the last year or so, I’ve seen countless articles, interviews, and even Facebook posts stating that political partisanship in the US is the highest it has been in decades. It seems that nearly every day there is a news story that – for one reason or another – compels people to “take a side,” which is often accompanied by a lengthy diatribe on social media. But as exhausting as these posts are, perhaps we can put them to good use. Maybe we could use the data to gain insights into a particular political party (or politician), or to summarize some of the leading issues we’re facing as a country. These ideas were the inspiration behind my second Capstone project.

The primary goal of this project was to use the Twitter feeds of 10 prominent politicians to build a model using both supervised and unsupervised machine learning techniques to predict the author – or political party – of unseen tweets. The secondary goal was to see if I could identify a coherent set of political topics using clustering and other topic modeling strategies.

The first step towards reaching these goals was to interface with the Twitter API and grab the most recent 3,200 tweets (the maximum allowed) for each of the 10 politicians / authors I wanted to classify. This dataset was then pared down to 2,100 tweets per author in the process of removing re-tweets from the sample. I wanted to get a representative sample spanning the political spectrum, so I chose the following prominent politicians: Donald Trump (of course), Hillary Clinton, John McCain, Bernie Sanders, Mike Huckabee, Barack Obama, Newt Gingrich, Mike Bloomberg, Kellyanne Conway, and Bill Di Blasio.

**Bag of Words & TF-IDF**

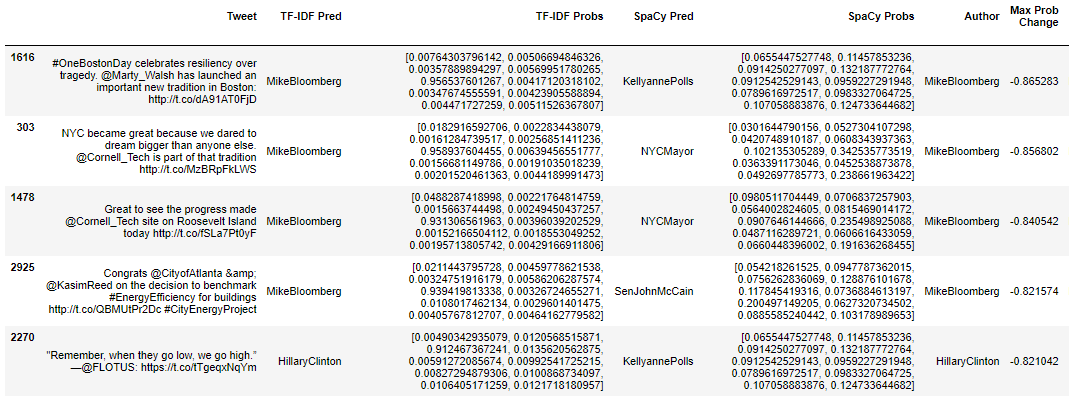
Next, I gathered, stored, and tokenized the tweets and created a standard bag-of-words dataset, which I used to train a Random Forest Classifier. To my surprise, the model was already 54.6% accurate. From there, I experimented with a Term Frequency – Inverse Document Frequency (TF-IDF) model, again using the Random Forest Classifier, and saw the accuracy jump to 64.1%. A gradient boosting model achieved 65.3% accuracy.

When fine-tuning my classification model using TF-IDF, I thought it would make sense to tokenize the tweet inputs using spaCy’s tokenizer, which would allow me to use lemmas rather than treating different versions of the same root word separately (is, was, etc. would all converge to “be” via spaCy’s tokenization). The callable function I built also removed hyperlinks, numbers, some common symbols (“&”), etc. The idea was that this would lead to marginally improved classification and/or topic clustering by reducing dimensionality and distances between words that mean the same thing.

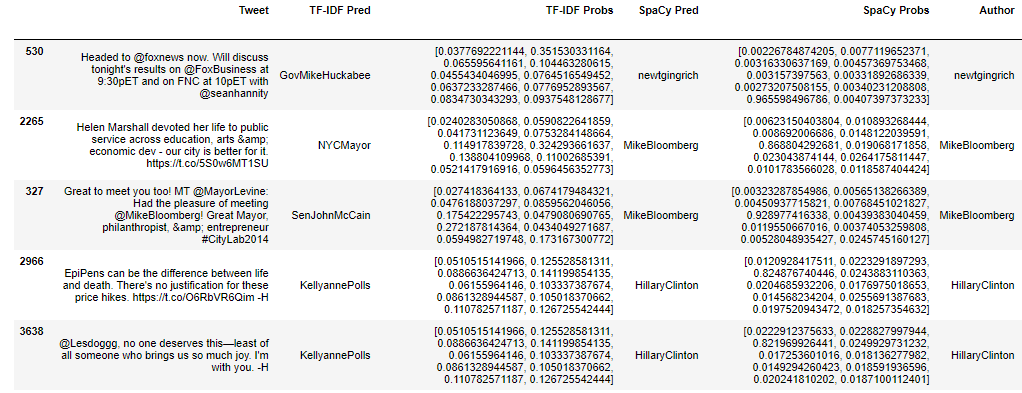
It turns out that this approach actually yielded slightly worse classification accuracy (64.0% vs 65.3% in the GBM models), and subsequent K-Means Mini-Batch clusters that were so similar to the default TF-IDF tokenizer that I could not definitively say that the clusters made more intuitive sense.

I also examined tweets that saw the largest magnitude changes (both in favor of and against the true author), but there was no discernable pattern with respect to what type of words/phrases/syntax led one model to perform better than the other.

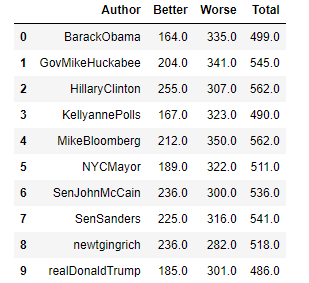
Examples where spaCY implementation was inferior:



Examples where spaCY implementation was superior:



Finally, I compared the probability scores of each model output, focusing on the probability assigned to the correct author. The standard TF-IDF tokenization approach fared much better on the test tweets here – see results below. “Better” or “Worse” refers to the relative performance of my customized spaCy tokenization versus the standard TF-IDF tokenization.



Hence I opted against the spaCy implementation of TF-IDF and returned to the default tokenization method.

**Centroids**

At this point I had a model that was 65.3% accurate on test tweets, and I began engineering some new features which made use of vector cosine similarity scores. To do this, I used my TF-IDF vectorized tweet data and calculated the “average” vector in the training set for each of the 10 politicians. I then calculated the cosine similarity score for each test tweet to the 10 average vectors and used those 10 features in new Random Forest and GBM models. This process is referred to as taking the “centroid” of the TF-IDF data. With these new features, the model accuracy had reached 70.0%. The next step was to try some unsupervised clustering and feature engineering to see if I could further improve the model’s performance.

**K-Means Clustering**

First, I tried various iterations (with varying numbers of clusters) of the K-Means Mini-Batch algorithm. The results were unstable, and I wound up defining 10 clusters with training data. Still, I tried to use the fitted model to cluster the test tweets, and use those predictions as features in the Random Forest and GBM models I had been testing so far. Unfortunately, the accuracy was essentially unchanged at 70.2%.

I have included the model statistics for reference.

Homogeneity: 0.164

Completeness: 0.192

V-measure: 0.177

Adjusted Rand-Index: 0.072

Silhouette Coefficient: 0.007

**Singular Value Decomposition & Latent Semantic Analysis**

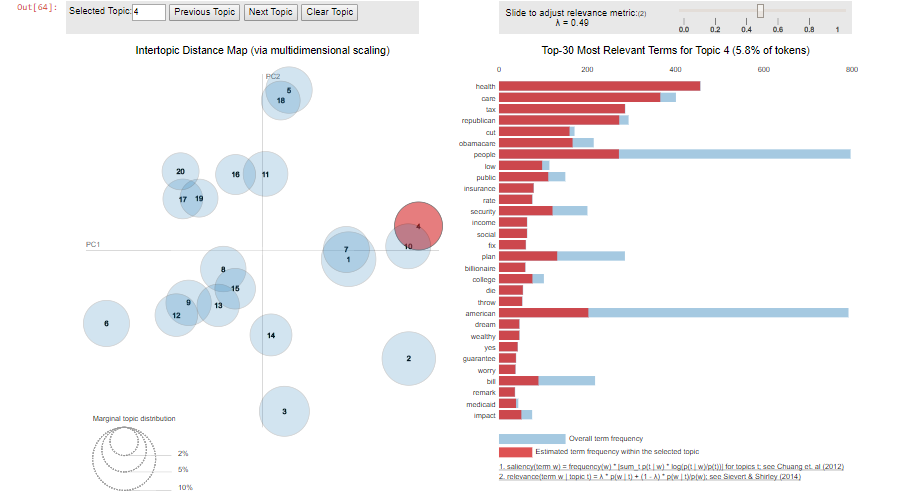
Next I tried a singular value decomposition (dimensionality reduction) on the same TF-IDF dataset I had been using (with the addition of the centroid features, as they were shown to boost model performance). I chose to reduce the feature space to 50 dimensions this time, though further analysis should be done to try to optimize the number of dimensions to project down into. The clusters here were similar in the sense of being written in the same style and with the same words (e.g. “live on @facebook”, “MAKE AMERICA GREAT AGAIN”, “joining us… “) but viewing them in the lens of “topics” was less viable. Of course, I still fit the training tweets to the singular value decomposition and normalization steps and transformed the test tweets to generate new, unsupervised features in the hopes of feeding back to the Random Forest and GBM classifiers I had used throughout the project. The results were very marginally better, with accuracy reaching 70.5% for the GBM model and 71.2% for a logistic regression model (new). I wondered, though, if there was a better way to extract meaningful clusters or topics from text – which led me to the Latent Dirichlet Allocation (LDA) algorithm.

**Latent Dirichlet Allocation (LDA) / Topic Modeling**

LDA is a really cool algorithm that makes two assumptions: 1) Words carry strong semantic information, and that documents discussing similar topics will use a similar group of words. Latent topics are then discovered by finding groups of words that frequently occur together within documents.

2) Documents are probability distributions of topics, and topics are probability distributions of words.

By passing tokenized tweet data and a pre-determined number of topics (this time I chose 20!) to this model, I was able to pull out some interesting insights from the data. Some topics are very coherent, organized, and intuitive - such as topic #4 (pictured below). The most relevant words include “health”, “care”, “tax”, “republican”, “cut”, and “obamacare.” Others are less clear - like topic #19, whose most relevant words are "life", "save", "student", "little", and "fox."



Just as I did before, I used the training set of tweets to fit the LDA model, getting a predicted topic score for each train and test tweet. Finally, I appended these predictions/features to the TF-IDF + centroid dataset, and ran several supervised ML classification models (including KNN, SVC, RFC, GBM, Logistic). Logistic Regression produced the best accuracy – 74.8%!

