Natural Language Processing Spring 2016 Final Project

PROJECT UPDATE

Did you meet your first milestone? Did you change your milestone?

Our initial milestone was to have rhetorical analysis for each debate and candidate completed by now. However, we decided to shift our goal to completing a basic TF-IDF vectorization for the candidates in order to put more focus on completing some predictive modeling. We were able to get significant work done towards this milestone.

What have you finished so far? Include background reading, development, brainstorming, and results.

We collected our data, and cleaned and parsed it in order to create a corpus with each candidate's speeches by debate accessible independently.

We copy and pasted all of our data (debate transcripts) into text files. We then wrote txt_to_json.py which uses regex expressions to separates debates into a dictionary of the form { "Speaker 1": { "Date 1": ["first answer", "second answer", "third answer"], "Date 2": ["first answer"] }, "Speaker 2": { } }. This dictionary contained moderators and other speakers, not just candidates. We stored this dictionary in a json file by election year, and election type (republican primary, democratic primary, or presidential election). Candidates were often referred to in multiple different ways so we wrote aliases.py which stores their different names in aliases.json. After running txt_to_json.py and aliases.py, we wrote and ran clean.py which removes all speakers from the debate who aren't candidates and aggregates candidate speeches so that their aliases are not considered different people. In order to run TF-IDF, we need parsed data, so we wrote parse.py which parses their text into words and additionally separates candidate speeches based on year and election type, which we need for win and loss labels. For example, "Barack Obama democratic 2008' and 'Barack Obama presidential 2008' become different people because wins and losses are on an individual election basis.

Once we compiled our corpus, we created context vectors for each candidate. We ran TF-IDF to vectorize a compilation of every debate for each candidate.

Because we have such little data, we also used dimensionality reduction in order to make our vectors more representative. Next, we labelled each candidate as winners (1) or losers (0) of the primary and/or presidential election the year that they ran. With these labels and the context vectors, we could run both a bag of words model and a kNN model.

For the bag of words model, we used a random forest classifier to use the vector features and labels to train a ‘forest’ of feature trees. We then ran our forest on the test data (which we also vectorized) in order to get the predicted winning and losing labels for each candidate as outputs. Because we did not choose for the classifier to run the same way each time, our results varied from around 76%-81% accuracy. Then, we ran our vectorized training set, trained labels, and test set through our k-nearest neighbors model, which also gave us an accuracy of 81%. We will discuss our results in more depth in the section below.

You should aim to have some results by the update. Describe them.

Our results are available in two ipython notebooks: random_forest_model.ipynb and k_nearest_neighbors_model.ipynb. When we ran these models on our test years, they both gave us an accuracy of 81%. While this sounds fairly impressive, nearly all of our train labels were 0, so this just meant that the majority of predicted labels were also 0. In fact, the kNN model assigned a value of 0 to every label, which makes sense conceptually. The random forest model tended to typically assign two or three labels as 1, in accordance to the actual labels. However, they were for the most part arbitrarily assigned by the model, due to the tiny dataset making it difficult to create a meaningful model.

Are the results satisfactory? If they aren't, what do you plan to modify/add?
Brendan

Both our KNN and random forest classifiers have the same problem. Firstly, because only one candidate can win each election approximately 80% of our data is losing candidates, making our labels extremely biased towards losses. This combined with the fact that overall we do not have sufficient data for either model, both strategies find the best strategy to be prediction almost entirely losses. Right now each document is a candidate year (meaning Obama in the 2008 democratic primary earned a win and separately Obama in the 2012 presidential election earned a win). This gives a total of 87 candidate year data points. Next we hope to get polling data for who won individual debates this will multiply our data by approximately 4x. Another step that could improve our results is to look at more interesting features other than just TF-IDF.

Updates to your project plan, like goals and techniques that have changed since your proposal.

The very first step we plan to take next is to label each candidate debate as a win or loss, rather than just label candidates as overall election winners or losers. We are planning on using gallup poll data to do this, if we can get access to their dataset through Gallup Brain. The reason we want to label by debate is because the way we are currently doing it, we simply have too few data points - only 53. With the addition of debate winners and losers to train off of, we will have significantly more data points. We also plan to weigh presidential election wins higher than primary debate wins, which will move us away from kNN modelling and towards other techniques, such as the random forest that we used.

Once we are able to complete this goal we will get a much clearer sense of what types of rhetorical patterns or features can help us in strengthening our model.

Difficulties or questions that I can help you with. If you're having code-related problems, note the filename and line numbers.

Our biggest question is simply what we can do about our seeming lack of data points. We are limited simply limited by the number of recorded electoral debates. Stemming off of this are there any features that may lend them well to our limited data set. Our other questions revolve around how to set several model parameters based on our data. Firstly, frequency threshold. We’re not sure how many words we should be trying to capture. We want to only capture defining words, however different candidates have vastly different amounts of text so a threshold that is too high may fail to capture the word usage of candidates with smaller corpuses. Once we have a good number of words what would be a reasonable range to reduce our dimensionality down to? Especially because of our limited number of data points we’re wondering what would be a reasonable range for our final vector dimensionality.