Dataset

Percentages by category:

1. Name Calling: 0.06734006734006734
2. Glittering Generalities: 0.12794612794612795
3. Testimonial: 0.04713804713804714
4. Plain Folks: 0.06060606060606061
5. Credit Claiming: 0.13804713804713806
6. Stereotyping: 0.020202020202020204
7. Slogans: 0.013468013468013467
8. Warmth: 0.013468013468013467
9. Patriotism: 0.12794612794612795
10. Repetition: 0.09427609427609428
11. Fear: 0.13804713804713806
12. Emotional Anecdotes: 0.08754208754208755
13. Bandwagon: 0.04377104377104377
14. Transfer: 0.020202020202020204

Logistic Regression

Constructing features

Since the logistic model needed a vector representation of the sentences that were training examples, I decided to use a bag of words featurization to establish this baseline. I first determined all the words in the corpus and then generated vectors that had a 1 in the indices of words that were in the training example.

Unfortunately, this creates a very large feature space.

Models and Experiments

I tried 2 types of logistic models:
First, I tried to use a logistic regression model that was k-class, and, with a hyperparameter grid search to find the the best learning rate, I was able to come across the best learning rate as being 1e-5. The training accuracy for this was 42%, using 5-fold cross validation.

I was able to compute the confusion matrix for the logistic regression model, which revealed several interesting confusions (for example, credit claiming and emotional anecdotes are often confused, which is reasonable since many instances of credit claiming involve telling emotional stories and then claiming credit for policies that helped these people).

I also tried to set up a binary classifier for each of the classes, and this resulted in extremely high validation accuracies, but the models did not have enough training data to compute anything real and were just guessing that the label wasn't present in all cases, resulting in the high accuracies.

LSTM + Pooling

Constructing features

The previous featurization lost information about the sequential nature of words, which is valuable in being able to determine characteristics about the persuasive techniques. I started by ranking the words in terms of how frequently they occur from largest to smallest. Therefore, the new sentence vectors were indices mapping to this list of words by their frequency.

Model architecture

The next architecture I set up had first an Embedding layer, which is responsible for converting the sentence vectors into a compressed representation of size 32. Then, there is a LSTM layer with 100 nodes, which is responsible for implementing the behavior that LSTMs demonstrate (logic gating that allows for previous samples to impact activation from the next timestep).

I next tried an architecture with a convolution and pooling layer between the input and the LSTM layer. I did this because convolution and pooling can often do a good job of heightening the impact of direct closeness between words in a sentence, which is valuable for certain kinds of categories, in this case.

Problems

Desperate need for more data for the model. Our feature space is too big - for logistic, should start considering alternate, more manual features maybe?

Consider starting with words and then finding other words that are like them. Manual featurizations might be able to make a difference.

Word Set Model

High Level

The goal of this model is to hit at the idea that many of these categories can be classified by a set of words that are used to trigger certain emotions. The underlying assumption is that it's better to use a word-cluster based model in this case, since a lot of the language used in these kinds of political speeches is very similar and very standard.

We have 3 datasets to mine from in this case - tagged examples from political speeches (not too many of these), full text of political speeches, and the full corpus of the Internet (including thesaurus-like databases), which is an in-the-wild representation of our language.

The motivation for this model is that, while it would be great to train some of the other models with a neural-style model, those models tend to be far more data-hungry. The number of tagged examples is small in this case, so the goal is to instead use those in collaboration with some kind of unsupervised form of learning based on other forms of text.

Tagged Clusters

The most direct form of word cluster is to identify clusters through finding the most frequent significant words used in the tagged examples that we do have. Then, we take the top k words that are most often used in the dataset. k is a hyperparameter here, and a tradeoff that's noticeable here is that there are a lot of words not used as frequently that are still very valuable to use. Keeping k big can help hedge back against overfitting to what's common in the speeches that have been tagged (and if the words really are infrequent and non-representative).

Cluster Expansion with Speeches dataset

Once we have core, tagged clusters, we can focus on expanding them by finding other instances in databases of untagged speeches (*not* the speeches from which the tags were already obtained). There are 2 ways of expanding the clusters:

1. Find examples of words around the central words in the full speech dataset and add those as part of the expanded cluster.

2. First, in the tagged examples, find sentences that contain the words from the sentence clusters. Then, find words that are some distance d from those words. Search for these words in the speeches database. Then, find the words around these words. We will add those in the set. An example might be if "happy" is a cluster word, we might have "a happy man" in the tagged sentence. Looking for "man" in the expanded dataset might find us a sentence "a jubilant man". We then get "jubilant" and add that to our expanded cluster.

Cluster Expansion with Wikipedia

This may be a way of adding to the dataset with a language set that is much larger than the speeches database. These examples are not specific to the problem at hand, so they can be downweighted when computing the $P(w|\text{expanded cluster for category c})$.

We perform the same techniques as for the full speeches database and downweight the results in the expanded cluster probabilities.

Classification

We now have core clusters and expanded clusters. We define α_1 and α_2 as the weight of each of the clusters and then define:

$$P(c|\text{word}) = \alpha_1 \cdot P(w|\text{core cluster for category c}) + \alpha_2 \cdot P(w|\text{expanded cluster for category c})$$

We will normalize as appropriate and treat α_1 and α_2 as hyperparameters, noting that $\alpha_1 > \alpha_2$, to weigh the contributions of the core cluster over the noisier expanded clusters.

Then, at test time, we will sentence tokenize and remove the insignificant words. Then, we define the score S where we have words w_i for $i = 0$ to n in a sentence of length n:

$$S_c = \frac{\sum_{i=0}^{n} P(c|w_i)}{n}$$

For each category, there will be a threshold β_C, which is a tunable or learnable hyperparameter, and if a sentence's score surpasses that threshold, it will be tagged as one of those categories. This allows us to access the multiclass categorization that we wanted as well.

Miscellaneous

- insignificant words - prepositions, articles, pronouns are excluded since they are the most commonly used words.