Debate Clustering

Daniel Fagerlie, Nicolas Salas, Sam Shinn
Western Washington University (WWU)
516 High Street, Bellingham, WA

***Abstract*— This project looks at how clustering can be used to classify sides of online debates within an arbitrary topic thread. For our tests, we will look at a pre-labeled dataset of people discussions within four topic threads: Gay Rights, Marijuana, Obama, and Abortion. We hope for each topic to be able to classify posts from these into 'for' or 'against' clusters. Hopefully this can serve as a first step towards being able to summarize large online conversations by classifying participant responses into various stances.**

Keywords— Feature Vector, Clustering, KMeans, Debates, Semantic Analysis, Python

I. Introduction

A. Research Motivation

On various social media platforms, such as Facebook, Twitter, or Reddit, there are certain personality figures or topics which draw thousands of participants into a single discussion. In these sorts of threads, it is not practical for someone to read every other contribution. Yet for someone to engage fully with the debate, they must give some consideration to all stances posted within the discussion. In any attempt to manuallyq read a significant portion of the discussion, an observer will quickly note a high degree of redundancy, that is, similar viewpoints (or "stances") expressed by different participants.

If these posts which express such similar stances could be classified and grouped together, a summarizing tool could then be applied to each grouping of posts. By this process we significantly reduce the amount of reading required to understand all stances represented in a discussion. Furthermore, this approach allows each participant's contribution to the discussion a degree of input power, which is to say that each contribution has at least some impact on the final summary, thus ensuring that every participant has a voice.

B. Abbreviations and Acronyms

ID	Identifier
PID	Parent Identifier
BOW	Bag of Words
TFIDF	Term Frequency - Inverse Document Frequency
SenAna	Method of creating feature vectors using most common nouns in the dataset
TFIDFSen	Method of creating feature vectors using TFIDF to select features from the dataset

C. Data

The data was retrieved from createdebate.com by researchers at UTDallas. The data consists of 4 different topics, which the curators consider inherently to be "stances" to which a participant may support or oppose. These "stances" (or topics) are abortion, Obama, marijuana, and gay rights. Each stance has a directory containing posts that either support or oppose the stance.

The posts are organized into two types of files, the posts data file and the meta file. The data file contains only the message from the poster. The meta file name is the same as the data file with the different extension (.meta). Associated meta files contains meta information about the message. This includes a unique ID of the post, and the parent ID (PID) of the post (which tells you which post this one is a response to). If the PID is -1, the post is not a reply to any posts. The meta data also includes whether the post supports or opposes the containing stance. Finally, the metadata includes a rebuttal which marks whether a reply to a post opposes the parent post.

Daniel Fagerlie, Nicolas Salas, Sam Shinn
Western Washington University (WWU)
516 High Street, Bellingham, WA

II. Methods

A. Overview of Framework

Our testing framework requires some flexibility in order to examine multiple approaches to clustering the posts of the various debates. Once the data is read into the program, the data is passed on to a module which will define a unique feature vector for the supplied topic thread. That feature vector definition is then used to calculate vector representations of each post in the topic thread. These vector representations can then be fed into a clustering algorithm, like KMeans, to be grouped into different classes.

By keeping things modular, we can interchange different feature vector modules in order to try a variety of feature vector representations. As a result, we are able to easily compare Bag Of Words (BOW), Term Frequency Inverse Document Frequency (TFIDF), Sentiment Analysis (SenAna), and a combination TFIDFSen representations.

Some options are available for clustering as well, though all are just minor variations on KMeans.

B. Preprocessing

To maximize possibilities for feature vector generation, we leave the post data as unedited as possible. Then which sort of tokenizer we use, or whether we modify punctuation, casing, etc,, things can be handled later by the feature vector implementation.

To simplify our dataset, we do make two adjustments by first omitting any replies found when reading in the data, and then by replacing any instances of multiple whitespaces, tab characters, or newlines with single whitespaces.

C. Feature Vectors

Feature vectors are handled in two stages: They are first defined by analyzing the entire collection of posts within a topic. Then feature vectors are calculated for each post using the defined feature vector.

Two feature vector implementations were used as baseline measurement: BOW and TFIDF. These feature vectors are defined by the top *N* words scored according to BOW or TFIDF algorithms, respectively, where *N* is some adjustable but arbitrary number of words. Calculation of these vectors is then simple word frequency for a given post. Preprocessing for these feature vector modules consists of removing punctuation, reducing all characters to lowercase, and using word tokenization.

The next tested module implementation is SenAna. This implementation uses word tags to identify nouns, and selects the top *N* most frequently occurring nouns in the topic to define a feature vector. The vectors are then calculated by first using the nltk sentence sentiment analyzer on each sentence, and then attributing the sentiment score of that sentence to any nouns occuring in the sentence and the feature vector definition. Preprocessing involves sentence tokenization and reducing characters to lowercase.

The last implementation we were able to test is TFIDFSen. This version uses TFIDF to identify words which define the feature vector and preserves the TFIDIF scores of each word to be used later as weight adjustments. To calculate the feature vectors, the nltk sentence sentiment analyzer is used on each sentence to determine their sentiment scores. For any word which is in the sentence and the feature vector, the sentiment score of the sentence is multiplied by the TFIDF score of the word and added to the word’s entry in the post’s feature vector. Preprocessing for this module involves word tokenization and removing punctuation for calculating TFIDF scores, sentence level tokenization for calculating sentiment scores, and reducing all characters to lowercase in both cases.

D. Clustering

To detect clusters among our feature vectors, we requisitioned the KMeans function from the SKLearn Python Library. Using this function we are able to pass our feature vectors in with a few parameters and return a list of cluster ID labels. The order of these ID labels correlates to the order of vectors which were passed to the function.

We have made a few options with regard to clustering, though as of yet they have not made much difference among any of the feature vector representations, and so the results of these options have not been formally analyzed or reported.

Nonetheless, these options are to normalize the feature vectors and/or remove any feature vectors (and corresponding posts) which are featureless. Normalizing the feature vectors makes all vectors length 1, except those vectors which were length 0, which remain length 0. Removing featureless vectors is done by detecting vectors of length 0 and removing them.

E. Evaluation

To evaluate the clustering method we used the adjusted_rand_score function provided by the SKLearn python library. The adjusted rand index gives a result between -1.0 and 1.0. A score of 1.0 would indicate that the clusters perfectly reflect the different stance label groupings and a "value close to 0.0 [indicates] random labeling independently of the number of clusters."[1]

We do not use measures of true-positive, true-negative, false-negative, and false-positive because of the nature of clustering. In practice, we do look at these values, but to say that "50% of the posts were correctly clustered" would be misleading, as that would be the minimum possible amount of "correctly clustered" posts. This is why we use the adjusted rand index: An ARI of 0.0 is no better than random selection, and an ARI of 1.0 is the goal of perfect clustering.

III. Results

Results were not as decisive as we had hoped. On average, we observed positive results for all feature vector implementations, though none were above ARI 0.05. TFIDFSen performed best (ARI 0.029), followed by BOW (ARI 0.015), followed by TFIDF (0.007), and finally SenAna (0.005).

[1] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html

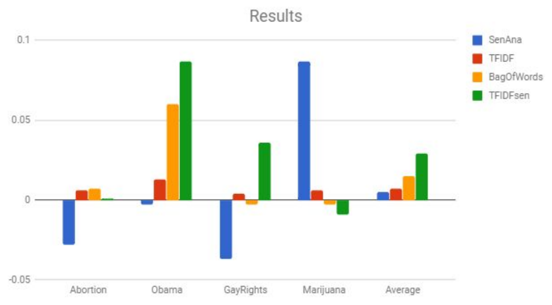

Figure 1 — results comparison of the indicated versions of the Feature Vector modules

IV. Discussion

Although we did not discover a feature vector model, when paired with KMeans, able to successfully cluster the data, we nonetheless view this endeavor as having great potential given more time. Our first reason for this is simply that our feature vector models did not explore all possible avenues of implementation, even among our own ideas. It is due to time constraints that we were unable to test a wider (and more complex) range of models.

Another reason we are not discouraged by the results is because the data might not have been well suited to the task; however, given the time constraint, it was the best we could find. Inspection of the dataset reveals far more than two viewpoints per topic. Since our goal is to cluster the posts into those different viewpoints, our evaluation is based on the binary stance labels assigned in the corpus, done by the researchers at UTDallas. It's hard to know how our classifier performed until we can study the corpus in more depth, or gather our own.

There are many avenues to explore for future work. The first and foremost is the data. If it could be curated for this task a bit more, certain patterns might reveal themselves easier than developing directly with the noisy real-world data. Of course, real-world examples would be used later when developing an application.

Second, there are many models for feature vectors which have not been explored here, which could make use of syntax trees, word embeddings, and/or clausal logic structures. Finally, KMeans was assumed to be an effective classification tool due to its flexibility: It can be applied to any number of clusters, and it does not require training data.

Daniel Fagerlie, Nicolas Salas, Sam Shinn
Western Washington University (WWU)
516 High Street, Bellingham, WA

However, there may be other options which are even better suited to this task.

V. Conclusion

During our work with feature vector representations, some had better performance than others. The ones that had the best performance were the more abstract methods involving sentiment analysis. Since this wasn't an exhaustive work within this area, further exploratory research is warranted. The primary focus should be on corpus quality and feature vector buildups. This points towards future work around word embedding and other abstractions that try to capture the essence of a debate stance.

References

Hasan, Kazi Saidul, and Vincent Ng. "Stance Classification of Ideological Debates: Data, Models, Features, and Constraints." Oct. 2013, www.hlt.utdallas.edu/~vince/papers/ijcnlp13-stance.pdf.