Project 2: Digital Democracy

Team: Richa Gadgil, Aniketh Bhat, Anand Rajiv

Part I: Clustering

For this part of the project we implemented a K-means clustering algorithm in order to classify utterances from a hearing into the committee that hearing was a part of.

K-means Clustering Algorithm:

First, we developed the k-means algorithm from scratch and compared it to SciKitLearn's KMeans function. Initially, we tested it with sample numeric data to ensure the accuracy of our algorithm and made changes based on those results. Once the cluster assignments were the same for both our k-means and sklearn's for the same k value, we moved on to testing it with feature vectors extracted from the text for each utterance. Even with large vectors with a lot of dimensions, after running a label comparison between our k-means and sklearn's, both still yielded the same cluster assignments.

One issue we faced was empty clusters. This usually happened when our feature vectors were ill defined. When the distances are so small, our k-means algorithm did not assign points to clusters accurately. To address this issue, we first took the largest cluster and assigned the furthest point in that cluster into the empty cluster. However, this took a lot of time, so we relied on more significant features.

Preprocessing:

Now that we know that our k-means algorithm works, we then focused on extracting significant features that would help optimize our clustering into different committees. The goal of preprocessing was to effectively filter out words which would confuse or

Some of the preprocessing we did to the text were:

- Removing punctuation (keeping only letters and numbers)
- Tokenizing words by whitespace using NLTK
- Tagged words by their part of speech and only kept nouns, verbs, adverbs, and adjectives

- Lemmatizing words so that words in third person are changed to first and words in different tenses are put into present tense

After performing the filtering and processing to the text, then we moved on to extracting our features. We returned the term frequency of each word in the utterance. However, one manipulation we did was to group similar words in each utterance together, and use that individual group and count the frequencies of each word that belongs in the group. To perform this task, we used wordnet to get generate synsets for each of the words and then using the synset id, we were able to group similar words together under that id. The key into the features dictionary was the id of the synset.

Feature Extraction:

For Feature Extraction, we attempted to discern what features would differentiate committee hearings. Based on our research, we determined some committees would be more descriptive than others and rely on more specific terms. A lot of this involved trial-and-error. Perhaps even more importantly, we had to figure out what didn't work -- we determined things like term frequencies, sentence length, etc. skewed vectors or relied on too much computational power.

Ultimately, our features were:

- Proportion of the hearing that was composed of Nouns
- Proportion of the hearing that was composed of Numbers
- Proportion of the hearing that was composed of Verbs
- Proportion of the hearing that was composed of Adjectives
- Proportion of the hearing that was composed of stop words

Vectorization and Results:

After returning the dictionary of features, we added the keys to a superset that contained all the keys across the entire corpus. This was used in order to determine the length of vector needed for the k-means algorithm. We constructed the sparse matrix by building the vector for each utterance and populating the dictionary with the keys that exist in the utterance. Then we passed in an np.array of lists where each list is the dictionary's values for the utterance.

We ran multiple iterations of k-means on the sample data for various values of k to determine the value for k that gives the best purity. A higher value of k gives us a better purity in general.

Experiments and Difficulties:

We experimented with a lot different feature extraction techniques and combinations of those techniques. At first we removed just stopwords and punctuation and generated term frequency from that. In an effort to improve the results, we compared lemmatizing vs. stemming and even both of them together, and we found that just lemmatizing the word gives best results. We also tried a different combination of parts of speech included such as just nouns, both nouns and verbs, and nouns, verbs, and adjectives. Instead of using term frequency, we also tried to work with just boolean values, and it didn't make much of a difference. We even tried to incorporate word length and sentence length into scaling the values of the vector, but it didn't make much of a difference to the purity scores. Ultimately, we stuck with proportions which would guarantee values would be between 0 and 1.

One of the biggest difficulties we faced was how to incorporate word similarity into the feature extraction. We knew that being able to group similar words together in a single utterance would make the features more significant, but we spent a lot of time experimenting on how we would implement it while keeping performance as efficient as possible.

At first, the keys in the feature dictionary were words. For each word in the utterance, we would get its first synset and then perform a wup similarity score for each key that exists in the dictionary. We set a threshold of greater than 0.5 to consider words "similar". If two words met this criteria, then a new key would not be created, and instead the frequency of the key that already exists would be incremented. As effective as this method is, it takes a heavy toll on performance as these operations are of order O(n^2). That's why we decided to rely on the synset ids as the keys into dictionary.

However, after implementing this, it didn't make much of a difference to the scores, so we decided to use as little features as possible while maintaining respectable scores to improve time constraints.

Results Visualized:
We generated a confusion matrix along with average F1-Score, precision, and recall for each cluster. This has been posted below.

```
My Kmeans labels:
K: 36
  1   0   1   3   0   0   0   1   0   0   1   0   0   0   1   0   1   2   0   0   2   0   0   2   0   0   0   0   0   1   0   0   0   0   0
  9  17  18  25  27   7  16  23   5  18  17  22   5  10  10   0  21   4  11  26   3  10  11   3  18  11   9   9   5  10   0   2   3   2   2
  0   5   2   8   7   3   0   3   2   6   8   1   0   0   1   0   9   1   0   2   2   0   0   0   5   2   0   0   1   2   0   0   1   1   2
  3  26  11  72  37  12  19  23  24  39  27  95  12   9  35   2  27   7  13  16   9  21   8  16  14   8   9  12  14  10   0   6   8  11  11
  2   2   3   3   4   1   2   8   3   3   5   1   1   0   4   1   4   2   1   4   2   0   1   1   2   2   2   0   0   1   0   1   1   2   0
  5  15   3  31  14  11   8   8  12  15  11   7   1   3   7   0   9   3   0   4   3   6   3   6   6   2   6   1   0   4   1   0   4   4   5
  1   2   0   9   2   5   1   2   3   2   2  10   0   1   2   0   8   2   1   7   3   1   0   2   0   0   1   1   4   1   1   0   2   0   0
  1  34   2  46   5   6   1   7   5  26  14  14   2   0   6   1  14   4   1   9   4   3   1   3   1   1   0   1   5   1   0   3   5   1   6
  2  29   0   6   9   2   3   0   3  14   9   4   1   0   4   1   6   1   0   0   1   1   0   1   0   1   0   0   2   1   0   1   2   0   3
  0   0   2   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   1   0   0   0   4   0   0   0   0   0   0   0   0   0   0
  0   0   0   1   0   0   0   0   0   0   1   0   0   0   0   0   1   1   0   0   0   0   0   0   0   0   0   0   0   1   0   0   1   0   0
  7   8  13  14  33   9  11  20  13  20  22  14   7   4   8   0  15   7   4  19   2   6   6   5  22  14   2   3   0   8   0   5   3   4   5
  1   0   1   1   1   0   0   0   2   1   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0
  4  19   5  27   8   6   6  15   6  17  13  16   5   0  14   1  10   3   1  10   7   5   1   5  14   3   1   0   2   5   0   0   1   2   1
  8   5  14  18  33   8  18  15  11  23  18  15   1  10  14   1  17  13   7  20   3  16  10  20  27   6   4   6   5   9   0   7   6   4   5
  9  17   4  39   4   3   0   5   6  30  15  68   2   1   9   0  15   5   1   1   5   4   1   3   2   1   1   1   8   2   0   0   5   0   1
  2   2   4  20  10   2   2   6   3   6  10  25   0   0   6   0   6   3   2   6   4   0   0   2   7   1   0   0   3   2   1   1   2   1   2
  1   0   0  11   0   0   1   4   3   1   1   1   0   0   0   0   1   0   0   1   0   0   0   0   1   0   0   0   0   0   0   0   0   0   2
  0   2   6   5   6   4   2   4   2   5   5   1   0   1   2   0   7   3   1   4   1   3   2   4   6   3   1   0   0   1   0   1   0   1   2
  5  22  14  39  20   7  14  20   8  30  24  44   4   3   9   1  26   5   4  17   5  12   4   5  16   7  11   6  10   4   2   1   7   2   1
  0   6   1   3   3   0   1   7   2   8   8  35   1   0   3   0  18   3   1   1   1   1   0   1   3   1   0   0   0   0   0   0   1   2   1
  5  22   3  57  13  10   3  18   4  15  20  22   1   1  11   0  19   7   4   6   6   3   4   3  10   2   0   4   5   8   0   0   5   3   6
  3   4   8  12  29   8   7   7  10  10   8   5   3   7   8   1  10   1  11  15   2   3   6  10  17   5   1   1   0   9   0   3   1   2   0
  2  33   2  61  18   4   5   8   9  29  19  48   4   0   9   0  21   3   4   4   4   6   0   8   3   2   1   1   8   2   0   3   4   3   2
  4  18  19  32  53  14  17  31  16  40  41  30   5  15  20   0  30  11   8  38   8  21  16   9  49  10  11   5   8  12   0   6   4   9   9
  0  11   0   6   0   0   0   0   1  11   0   2   0   0   0   0   4   1   0   0   0   0   0   0   1   0   0   1   2   4   0   0   0   1   0
  4  12  26  59  42  12  30  33  25  37  52  43   6  11  35   2  41   9  13  30   8  34  17  15  46  20  20   6   6  13   1   6  11   8  10
  7  11   8  31  13  15   4  18  16  20  17  25   2   6  13   1  17   7   2  17   2   4   1   6  13   7   4   2   1   3   0   2   6   4   4
  4  13   8  32  25   6  12  21  11  24  19  32   4   7  14   0  24   4  10  19   5   5   9   8  17  11   4   7   3  10   1   4   1   3   7
  3   0   5   6   4   2   1   1   5   8   5   4   0   0   5   1   1   1   2   2   4   2   0   1   4   3   3   2   0   2   0   1   1   1   0
  0   2   3  30  20   1   2   6  16  46  27  15   0   1   3   0   8   5   0   9   5   3   0   0   4   1   1   0  12   1   0   1   1   2   1
  8   3   5  20   5   6   1  11   5   6  17   0   1   1  14   1   9   6   4   6   0   0   3   0   2   1   0   0   0   5   0   1   1   1   1
  2   7   1  39   3   2   3   3   4  16   4  30   1   0   4   0   6   6   1   0   2   2   1   3   3   1   2   0   1   3   1   1   2   3   1
  7  18  18  39  29  17  31  27  12  25  33  30   6   8  13   2  20   7  15  34   8  18  10  11  38  14  16   9  11  12   0   6   8   4   1
```

Part II: Decision Trees

For this part of the project, we implemented a decision tree in order to classify utterances from hearings into the committee that the utterance was a part of.

Preprocessing and Feature Extraction
The significant features being extracted are the same ones that we extracted for our clustering algorithm. As we stated above, the features were selected through numerous iterations of trial and error.

Difficulties:
A big difficulty with this program was figuring out how to programmatically represent the tree. In the end, I decided to represent it as a dictionary where the key is the question(variable <= split_point), and the value is a list where the first value is the portion of the tree branching off when the answer is yes, and the second value is the portion of the tree branching off when the answer is no.

Results:
These were the results when I ran the decision tree algorithm with 5000 rows (~17% of the dataset). When splitting the data into training and testing sets, I split it so that 80% of the data will go into the training set, and 20% of the data will go into the testing set.

Committee Name	Precision	Recall	F1 score
Housing & Community Development	0.208	0.3	0.246
Public Employees	0	0	0
Arts, Entertainment	0.063	0.077	0.069
Health	0	0	0
Judiciary	0.218	0.144	0.173
Human Services	0.037	0.037	0.037
Privacy & Consumer Protection	0.039	0.028	0.033
Governmental Organization	0.1	0.033	0.05
Insurance	0.13	0.27	0.178
Agriculture	0.146	0.048	0.072
Appropriations	0.088	0.069	0.078

Overall Accuracy: **0.14**

Part Three: SVM Accuracy Score

SVM works by splitting up the data using hyperplanes. The goal is to find multiple hyperplanes in N-dimensional space to split up the data points, and classify the data points on each side. It should be created where there is a maximum margin between the two different classifications of data points.

Using the SVC module imported from scikit learn, we were able to reach an accuracy of as much as 57%.

Among other methods we tried were Naive Bayes and Logistic Regression. Although Naive Bayes was faster in terms of speed, both were not able to yield the same amount of accuracy as SVM.

First, we vectorized the data and removed stop words using TF-IDF Vectorizer from scikit learn. We were able to pass this directly into the SVM module we had previously set parameters for. We also set the gamma to auto to take the expected furthest margin value.

Initially, we had trouble deciding what type of kernel to use. Each kernel generates a different kind of hyperplane between the clusters.

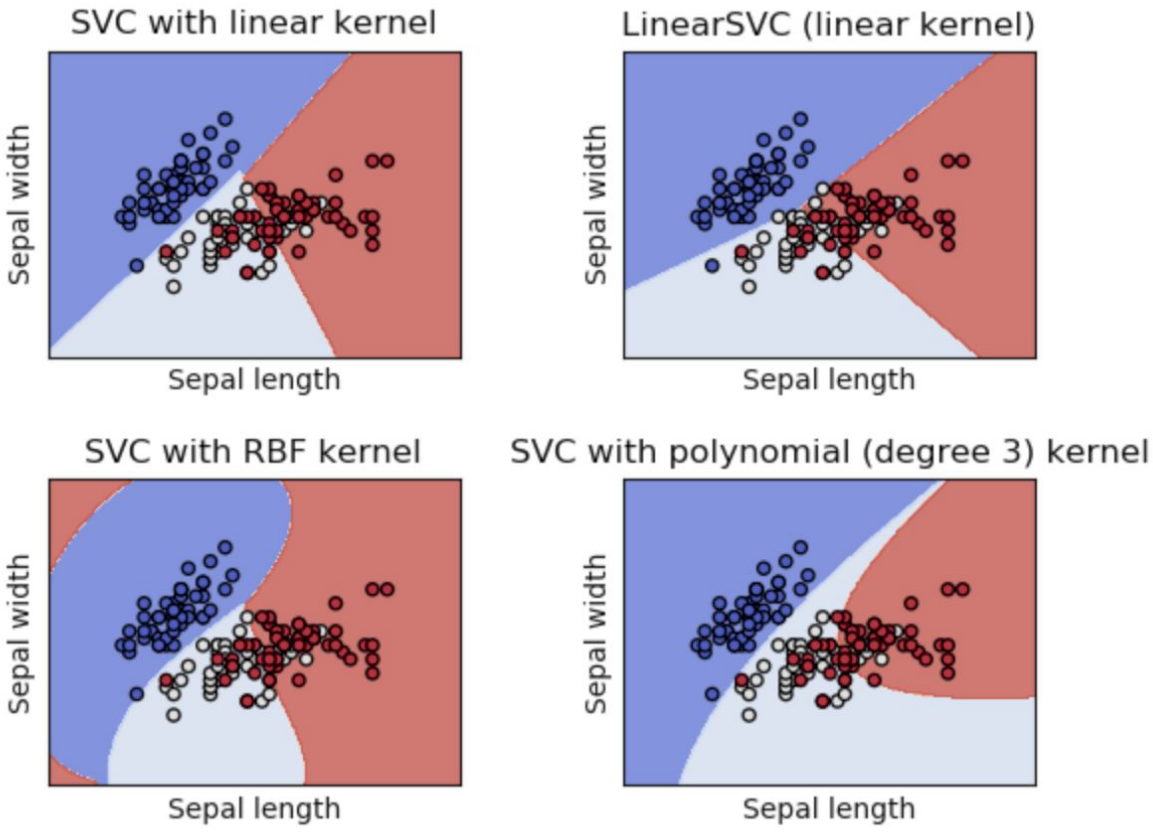

Photo courtesy of SciKit Learn. [https://scikit-learn.org/stable/modules/svm.html]

Ultimately, we decided on Linear it provided the best score and intuitively we could see each vector occupying its own space in the n-dimensional field. To have “rbf” and other vectors could lead to cross-over hyperplanes.

Ultimately, we were able to get:

Overall accuracy: 0.57
Average Precision: 0.49
Average Recall: 0.59
Average F1 Score: 0.51