1. **Discussion of data format**

I downloaded the dataset from Kaggle to my personal laptop and unzipped the folder. The dataset had a large size, so it took a long time to unzip the folder. The folder was saved on my desktop and I opened the jupyter notebook and got a direct access the data. From the JSON files, I thought that working with ‘body\_text’ would be much meaningful. I used the glob module and glob function to access all the JSON files from different folders. I used two notebooks for this project: the first notebook had all my work for clustering and the second notebook had all my work for sentence ranking or summarization part.

On the first notebook, I used the 10 percent of the overall data and the total size of these data was 4,594 texts that had been collected from the ‘body\_text’ of 4,594 files. I used a data frame to store these texts after opening the JSON files using the ‘load’ function from JSON library and looping through the file. The data frame was consisted of one column and the value of texts, which were objects and they were listed in rows.

On the second notebook, which was where I had ranked sentences, I used only 0.001 percent of the total file, which was 99 files in total, to save running time. I used the same approach as the first notebook to get texts from the body\_text.

1. **Discussion of tokenizer**

On the first notebook for clustering, I took these texts from rows into a list, which a text from a ‘body\_text’ from a file would be stored as a string in an array. Then I began the preprocessing step where I used nltk to remove special characters, digits, and stop words after word tokenization. The cleaned document was normalized, and it took the list of my data. I used the nltk library for word tokenizing, regular expression library to clean digits and special character, the ‘lower’ function to change all string characters into lower case character and strip to remove characters before and after the string to make it cleaner. Then I used a TfidfVectorizer to transform the data. I used a parameter of ngram\_range (1,2) in my vectorizer which means my minimum feature would be 1 and a maximum feature would be 2 while feature extracting.

On the second notebook for sentence ranking, I stored these texts into a list of strings and split the strings by comma and used the ‘extend’ function to extend the list by adding all strings of the texts. The size of the very first list that I created after storing the texts from the JSON files was 99 while the size of the second list after the split was 918, containing all the sentences in a single array. I had tried preprocessing the data I was working with, but the sentences were not well structed when I applied the ranking algorithm so I decided to work with the original data. Then I used CountVectorizer with no parameter and used fit\_transform to vectorize my data.

1. **Discussion of clustering method**

On the first notebook, after vectorizing I performed a Silhoutte test to help me determine the better number of clusters for the k-means clustering method I was going to use. Hence, I imported ‘silhouse\_score’ and looped through different numbers of clusters from 2 to 15 with k-means model and fitting the tf-idf matrix. The silhouse\_score function the tf-idf score matrix and the k-means fitted model as parameters to compute the score for different numbers of clusters. Based on the results, I found that 11 clusters seemed better to categorize the data. The score was 0.022, and though this was one way of determining the number of clusters, the score was very far from 1, which means if there would be a better number of cluster if I had a larger range for number of clusters. Then I built my k-means model using that number of cluster, with a maximum iteration of five, and n\_init (the number of the k-means algorithm would be run) was five which was below the default to save model running time, and a random\_state (the seed used by the random number generator of ten. After running my model, I printed the key terms and sentences for eleven clusters.

I had seen that there were numbers and special characters in my clusters, and I am not sure how that happened. Also, I saw that there were non-English words. Though I did not specifically do text cleaning to avoid non-English words, which could have been done in preprocessing part, I doubt it would make a difference as the preprocessing method did not prevent numbers and characters from being in the clusters.

1. **Discussion of ranking (summarization)**

After vectorizing the data using CountVectorizer, I calculated sentence similarity using cosine similarity. Then, I created a similarity matrix that is a product of the similarity array and its transpose. This matrix was passed to the networkx library’s function called ‘from\_numpy\_array’. Then passed the result to the pagerank algorithm to rank the sentences. Then I applied this algorithm to my data and sorted the sentences based on their ranks.

**Resources/Contributors:**

<https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarization-textrank-python/>

Lecture Code by Dr. Christian Gran on Basic RecSys and Clustering Lecture (cs5292sp20class24.py)

Keerti Banweer

<https://www.geeksforgeeks.org/silhouette-index-cluster-validity-index-set-2/>

<https://kanoki.org/2018/12/27/text-matching-cosine-similarity/>