Università della Svizzera italiana

Faculty of Informatics

May 10, 2018

Data Analytics Project

Clustering Texts: Report

Sumeet Gyanchandani

1 Dataset

The dataset contains text from 3 different sources, namely, Enron emails, NIPS papers, and Kos blogs. For each text collection, D is the number of documents, W is the number of words in the vocabulary, and N is the total number of words in the collection. After tokenization and removal of stopwords, the vocabulary of unique words was truncated by only keeping words that occurred more than ten times.

2 Pre-processing the data

The dataset that I received was already converted into lowercase, the Stopwords, Punctuations, Spaces were removed and even Stemming was done. Hence, there wasn't much requirement of data cleaning.

The vocabulary and docword files for all the collections were loaded in R. Further, these pairs of vocabulary and docword files were merged by wordID.

Strictly for exploratory purposes, the collections were aggregated on the basis of words. This helped us get the top 10 words for each collection.

3 Exploration

The important dimensions of these collections are:
Enron Emails:
D=39861
W=28102
N=6,400,000 (approx)

NIPS full papers:
D=1500
W=12419
N=1,900,000 (approx)

KOS blog entries:
D=3430
W=6906
N=467714

I started with basic plots of top 10 words in each collection and then went into sentiment analysis.[1]

3.1 Top 10 Analysis

The following are the Top 10 words in their respective collections.

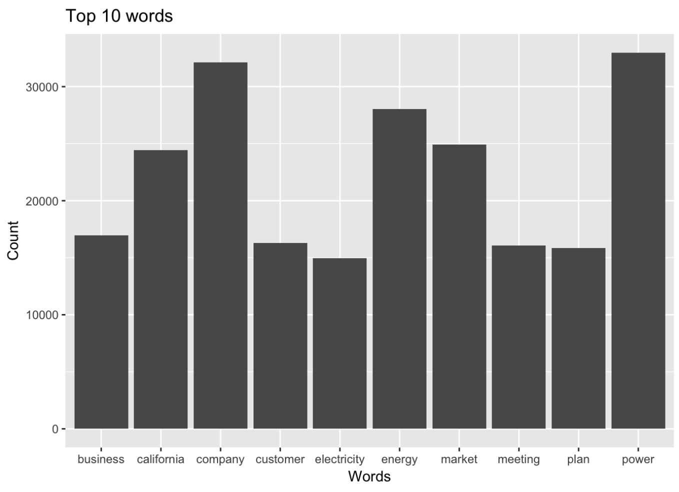


Figure 1. Term Frequency Histograms for Enron Emails

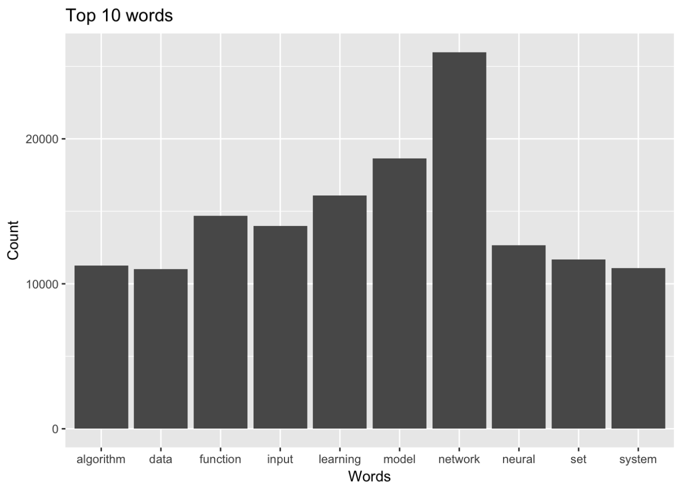


Figure 2. Term Frequency Histograms for NIPS data

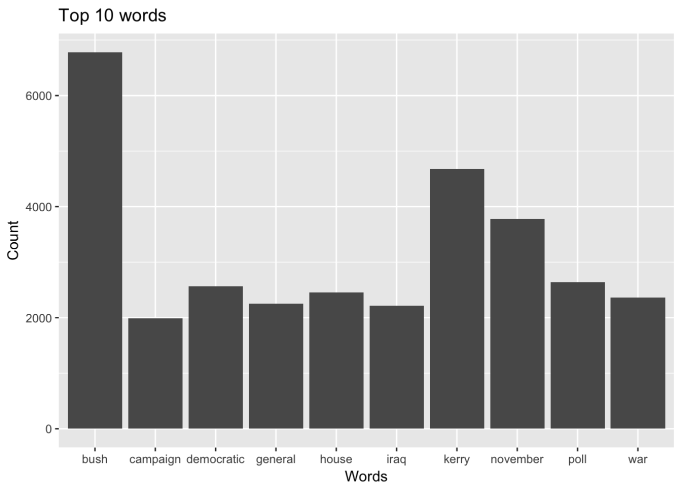


Figure 3. Term Frequency Histograms for KOS data

3.2 Sentiment Analysis

One of the datasets that I was given originally belonged to Enron Corporation. You see, the Enron scandal that came to light in 2001 which eventually led to the bankruptcy of the company. This is the largest corporate fraud that had happened so far. The Enron top-honchos used what is called Mark-to-market accounting to make up their financial statements. They used this accounting and financial shenanigan to trick investors and make money which later went uncontrollable and finally led to the bankruptcy of the firm. There is also a documentary regarding this, which I have mentioned in the references. [2]

Right after the scandal, emails exchanged between the Enron executives were made public. I chose to perform sentiment analysis on the Enron collection because finding out the sentiments of Enron executives when the scandal was taking its shape could have alerted the stakeholders and intelligent agencies regarding the fraud.

Once the dataset was loaded and ready for analysis, I used the R package **tidytext** to extract sentiment from the word list. The tidytext package comes in three different lexicons:

- AFINN
- nrc
- bing

A lexicon is a set of vocabulary of a particular genre. For example, the AFINN lexicon assigns 2476 pre-defined words with a score of -5 to 5 where negative score means negative sentiment and vice versa. The nrc lexicon assigns 13901 predefined words into one of the 10 different sentiments viz positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise and trust. Whereas, bing lexicon categorizes 6788 predefined words into either positive or negative sentiment.

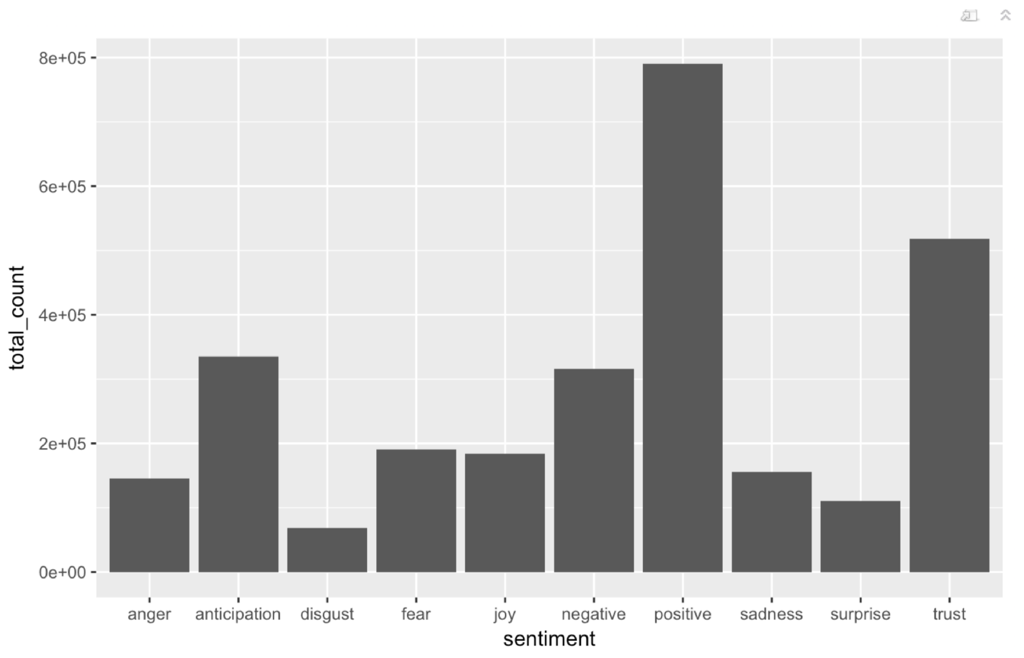


Figure 4. Sentiment Frequency Histogram

From the above sentiment bars, it is evident that they had a very positive sentiment among themselves. This could be because they were successful in fooling the stakeholders throughout that period. Another sentiment that got a place into the top sentiments was **anticipation**, which is again quite obvious as those executives were afraid of being caught.

Notice the dip in the positivity of the sentiments between the document ID 25000 and 35000 in Figure. 5. This phenomenon is worth analyzing to find out the exact reason behind this negative sentiment. With the help of this analysis, one can pinpoint the exact documents that could lead to evidence of fraud during the investigation.

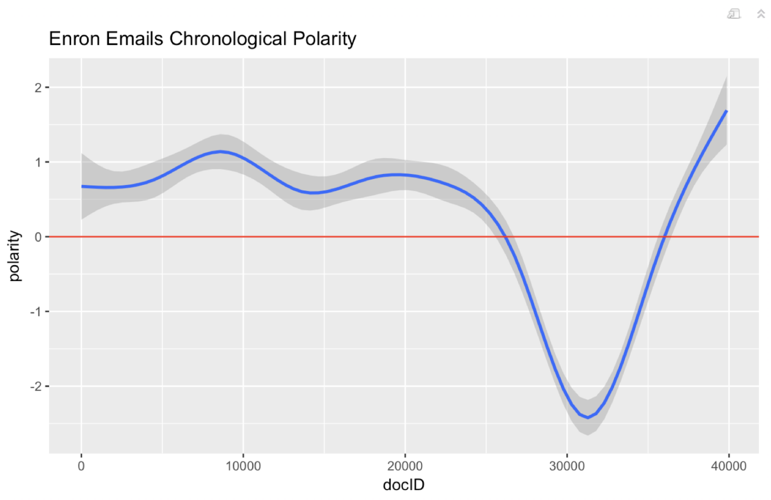


Figure 5. Sentiment Frequency

The following figure is a frequency analysis of the sentiment words.

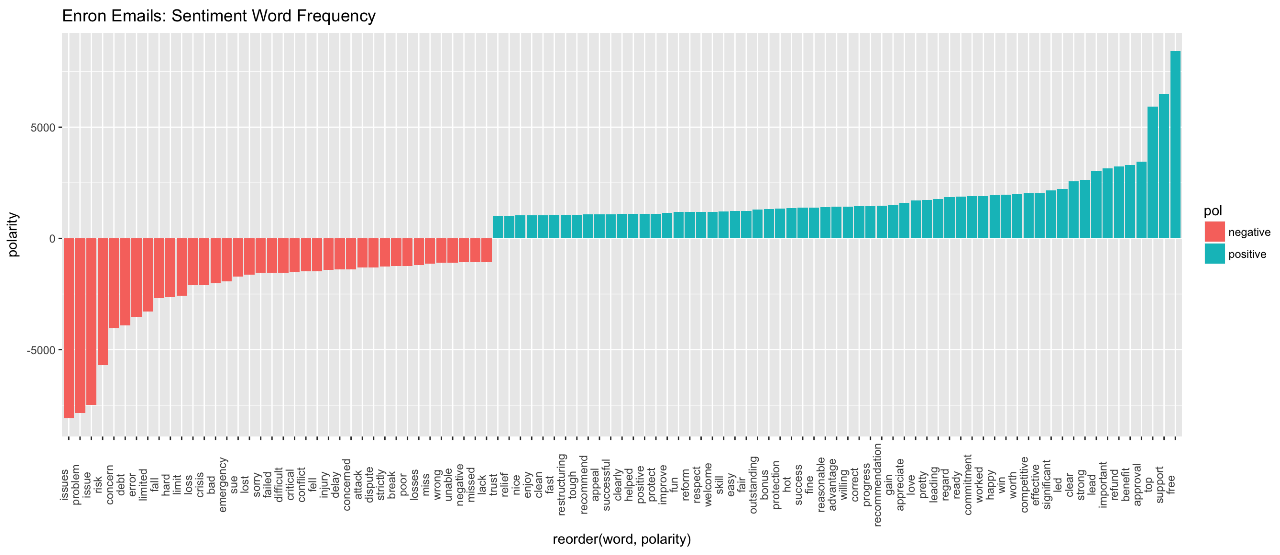


Figure 6. Sentiment words

Figure 7. shows Word Cloud of the words that the Enron executives spoke about the most and Figure 8. shows a Radar Plot of the top 8 sentiments from the analysis. It shows pretty the same conclusion that we drew above.

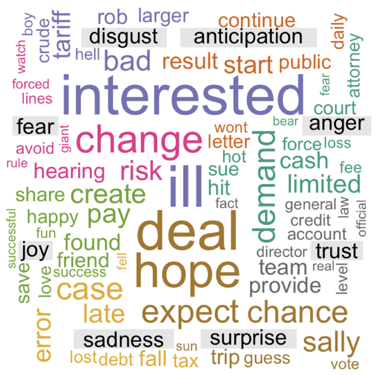


Figure 7. Word Cloud

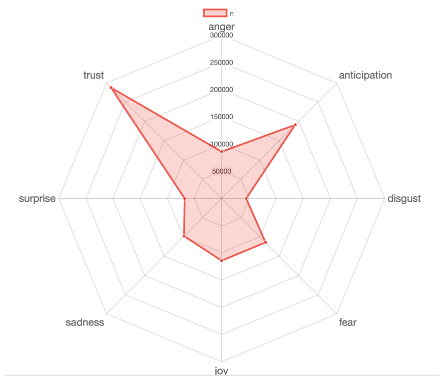


Figure 8. Radar Chart

4 Clustering Algorithms

Before I started clustering I had to perform some more pre-processing steps in order to make the data ready for the clustering algorithms. [3] These steps are:

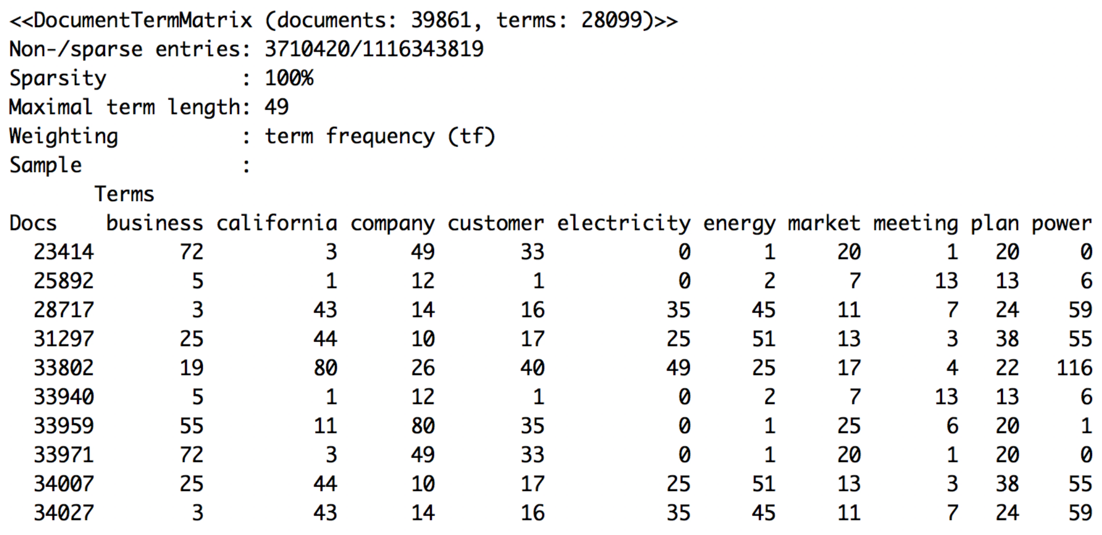
1. Document-Term Matrix (DTM)
2. Term Frequency-Inverse Document Frequency (TF-IDF)
3. Distance Matrix

4.1 Document Term Matrix (DTM)

The Document-Term Matrix (DTM) is a matrix that lists all occurrences of words in the corpus, by document. In the DTM, the documents are represented by rows and the terms (or words) by columns. If a word occurs in a particular document n times, then the matrix entry for corresponding to that row and column is n, if it doesn't occur at all, the entry is 0.

We can easily get the Term-Document Matrix (TDM) by transposing the DTM. We can use any of these, but I'll use the DTM throughout my analysis.

Note: The size of the DTMs were very large, hence I didn't upload this to the Dropbox Project Folder. It is clear that the large majority of words will appear only in a few documents. As a result, my DTMs are sparse, i.e., a large number of its entries are 0.



```
<<DocumentTermMatrix (documents: 39861, terms: 28099)>>
Non-/sparse entries: 3710420/1116343819
Sparsity           : 100%
Maximal term length: 49
Weighting          : term frequency (tf)
Sample             :
       Terms
Docs    business california company customer electricity energy market meeting plan power
  23414       72          3      49       33           0      1     20       1   20     0
  25892        5          1      12        1           0      2      7      13   13     6
  28717        3         43      14       16          35     45     11       7   24    59
  31297       25         44      10       17          25     51     13       3   38    55
  33802       19         80      26       40          49     25     17       4   22   116
  33940        5          1      12        1           0      2      7      13   13     6
  33959       55         11      80       35           0      1     25       6   20     1
  33971       72          3      49       33           0      1     20       1   20     0
  34007       25         44      10       17          25     51     13       3   38    55
  34027        3         43      14       16          35     45     11       7   24    59
```

Figure 9. Document Term Matrix for Enron Emails

4.2 Term Frequency-Inverse Document Frequency (TF-IDF)

In the above DTM, you will notice that terms that occur frequently have a high value associated with it. However, a term's high frequency within a document means little, if that term appears frequently in other documents in the corpus. In other words, terms that occur frequently within a document but not frequently within the corpus receive a higher weighting as these words are assumed to contain more meaning in relation to the document. A problem with scoring word frequency is that highly frequent words start to dominate in the document, but may not contain as much "informational content" to the model as rarer but perhaps domain-specific words.

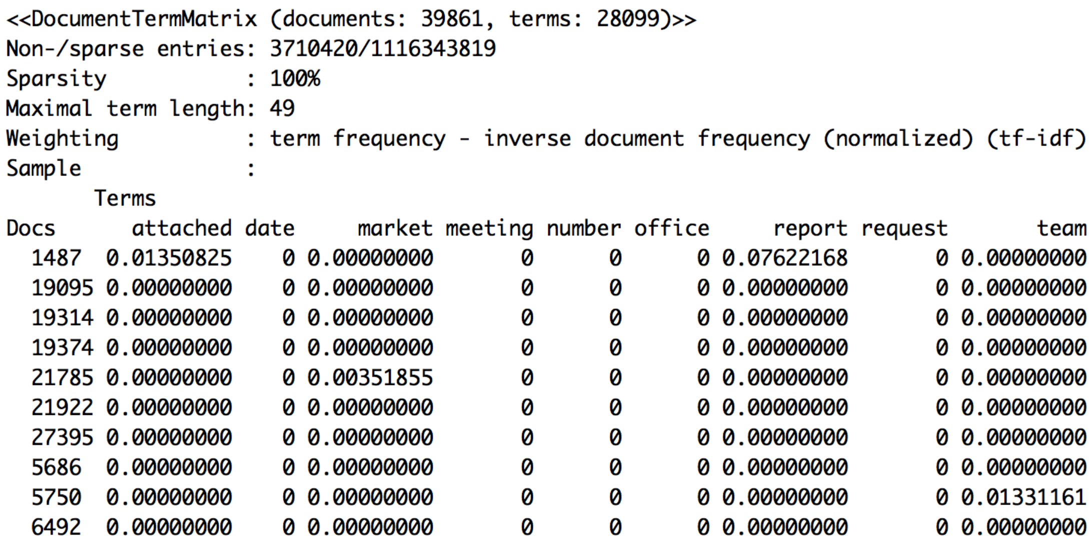
One approach is to rescale the frequency of words by how often they appear in all documents so that the scores for frequent words like "company" that are also frequent across all documents are penalized.

This approach to scoring is called Term Frequency-Inverse Document Frequency (TF-IDF) where:[4]

Term Frequency: is a scoring of the frequency of the word in the current document.

Inverse Document Frequency: is a scoring of how rare the word is across documents.

The scores are a weighting where not all words are equally as important or interesting. They highlight words that are distinct (contain useful information) in a given document.



```
<<DocumentTermMatrix (documents: 39861, terms: 28099)>>
Non-/sparse entries: 3710420/1116343819
Sparsity           : 100%
Maximal term length: 49
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
Sample             :
       Terms
Docs     attached date     market meeting number office     report request       team
  1487 0.01350825    0 0.00000000       0      0      0 0.07622168       0 0.00000000
  19095 0.00000000   0 0.00000000       0      0      0 0.00000000       0 0.00000000
  19314 0.00000000   0 0.00000000       0      0      0 0.00000000       0 0.00000000
  19374 0.00000000   0 0.00000000       0      0      0 0.00000000       0 0.00000000
  21785 0.00000000   0 0.00351855       0      0      0 0.00000000       0 0.00000000
  21922 0.00000000   0 0.00000000       0      0      0 0.00000000       0 0.00000000
  27395 0.00000000   0 0.00000000       0      0      0 0.00000000       0 0.00000000
  5686  0.00000000   0 0.00000000       0      0      0 0.00000000       0 0.00000000
  5750  0.00000000   0 0.00000000       0      0      0 0.00000000       0 0.01331161
  6492  0.00000000   0 0.00000000       0      0      0 0.00000000       0 0.00000000
```

Figure 10. Term Frequency-Inverse Document Frequency Weighting

Word clouds of the collections post TF-IDF Weighting:

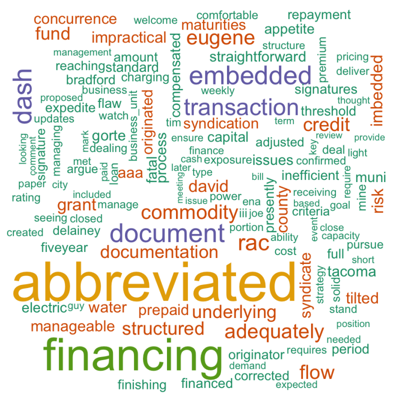


Figure 11. A word cloud from the Enron collection post TF-IDF

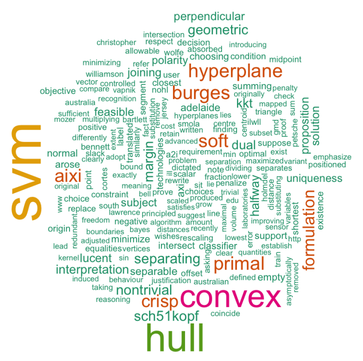


Figure 12. A word cloud from the NIPS collection post TF-IDF

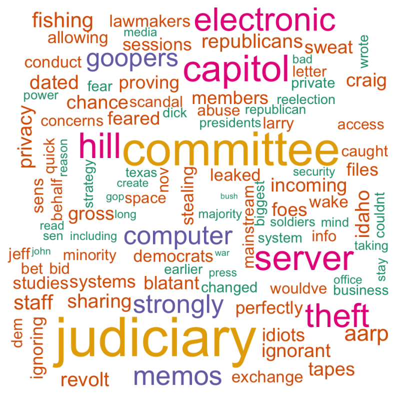


Figure 13. A word cloud from the KOS collection post TF-IDF

4.3 Distance Matrix

Next, I calculated the distance matrix between the documents. This distance is what the clustering algorithms will use to cluster documents. The measure that I used was **Cosine Similarity**:

Given two vectors of attributes, A and B, the cosine similarity, $\cos(\theta)$, is represented using a dot product and magnitude as:

$$\text{cosine similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||\,||\mathbf{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

where, A_i and B_i are components of vector A and B respectively.

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures cosine of the angle between them. The cosine of 0^o is 1, and it is less than 1 for any other angle in the interval $[0, 2\pi)$. It is thus a judgment of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors at 90^o have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude. [5]

In simple English, Cosine similarity is generally used as a metric for measuring distance when the magnitude of the vectors does not matter. This is true for our case, as we are working with text data represented by word counts. We could assume that when a word occurs more frequent in document 1 than it does in document 2, that document 1 is more related to the topic. However, it could also be the case that we are working with documents of **uneven lengths**. Then, the word probably occurred more in document 1 just because it was way longer than document 2. Cosine similarity corrects this.

4.4 Clustering Algorithm 1: Hierarchical Clustering

The first algorithm that I used is Hierarchical Clustering. [6] There are two types of hierarchical clustering strategies:

Agglomerative: where we start out with each document in its own cluster. The algorithm iteratively merges documents or clusters that are closest to each other until the entire corpus forms a single cluster. Each merge happens at a different (increasing) distance.

Divisive: where we start out with the entire set of documents in a single cluster. At each step, the algorithm splits the cluster recursively until each document is in its own cluster. This is basically the inverse of an agglomerative strategy. [3]

The function that we'll use is *hclust* which does agglomerative hierarchical clustering. Here's a pseudo code of how it works:

1. Assign each document to its own (single member) cluster
2. Find the pair of clusters that are closest to each other and merge them. So you now have one cluster less than before
3. Compute distances between the new cluster and each of the old clusters
4. Repeat steps 2 and 3 until you have a single cluster containing all documents

Interpretation:

As the entire corpus would have been very difficult to visualize. I chose a small sample for my analysis. The deductions may or may not scale to the entire corpus. As we work our way down the dendrogram, each branch point we encounter is the distance at which a cluster merge occurred. Clearly, the most well-defined clusters are those that have the largest separation; many closely spaced branch points indicate a lack of dissimilarity (i.e. cosine similarity, in our case) between clusters. Based on this, the figure reveals that there are 7 well-defined clusters in our Enron emails sample.

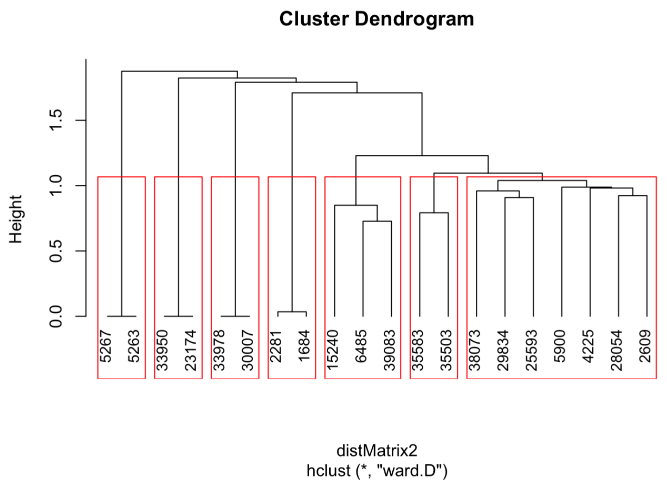


Figure 14. Dendrogram from Hierarchical Clustering of Enron Emails

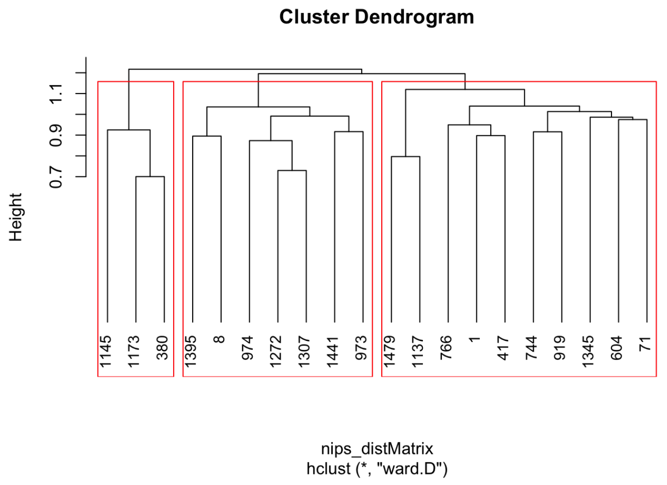


Figure 15. Dendrogram from Hierarchical Clustering of NIPS data

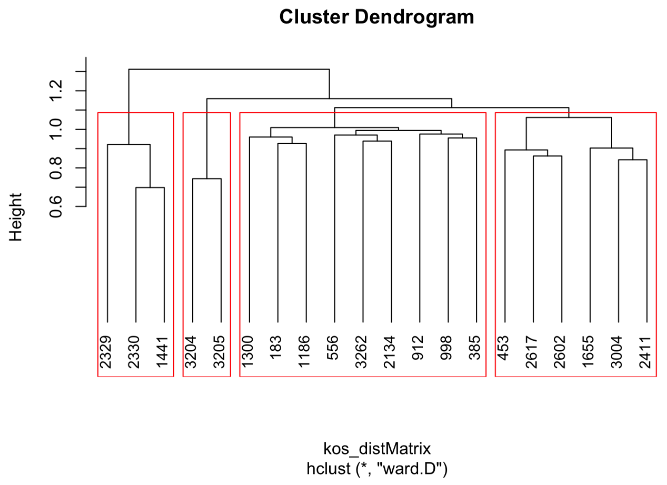


Figure 16. Dendrogram from Hierarchical Clustering of KOS data

4.5 Clustering Algorithm 2: K Means Clustering

In hierarchical clustering, I did not specify the number of clusters upfront. This was determined by looking at the dendrograms after the algorithm had done its work. In contrast, our next algorithm, K Means, requires us to define the number of clusters upfront. The algorithm then generates k document clusters in a way that ensures the within-cluster distances from each cluster member to the centroid (or geometric mean) of the cluster is minimized.

Here's a pseudocode of the algorithm:

1. Assign the documents randomly to k bins
2. Compute the location of the centroid of each bin
3. Compute the distance between each document and each centroid
4. Assign each document to the bin corresponding to the centroid closest to it
5. Stop if no document is moved to a new bin, else go to step 2

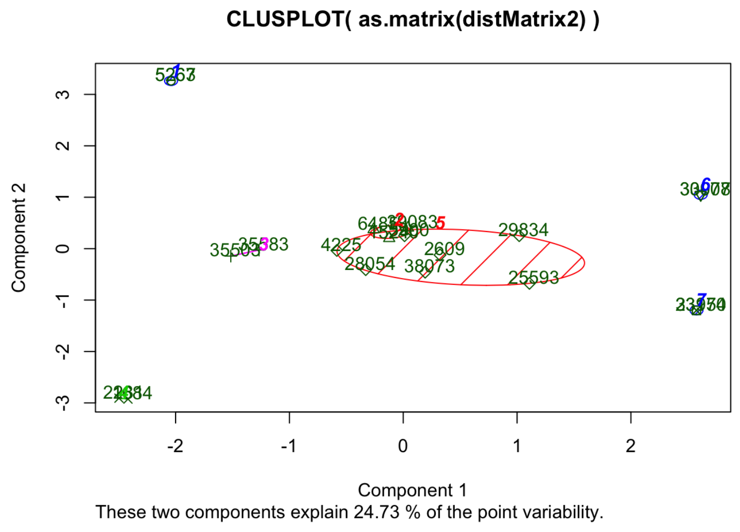


Figure 17. K Means Clustering for Enron Emails

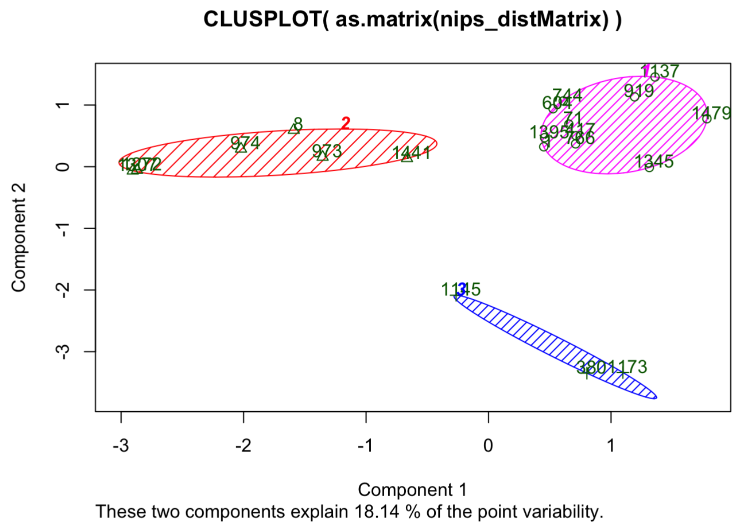


Figure 18. K Means Clustering for NIPS data

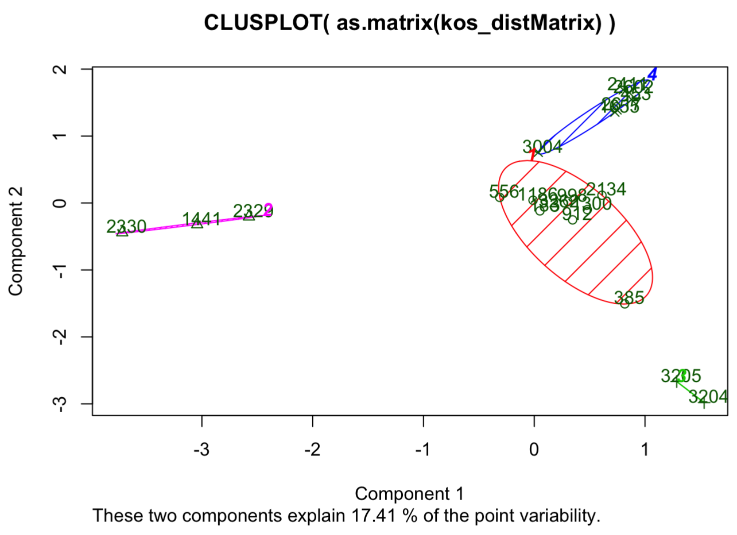


Figure 19. K Means Clustering for KOS data

Interpretation:

The cluster plot shown in the figure above needs a bit of explanation. As mentioned earlier, the clustering algorithms work in a mathematical space whose dimensionality equals the number of words in the corpus. Clearly, this is impossible to visualize. To handle this, mathematicians have invented a dimensionality reduction technique called Principal Component Analysis which reduces the number of dimensions to 2 (in our case) in such a way that the reduced dimensions capture as much of the variability between the clusters as possible.

5 Conclusion

5.1 Comparison of the algorithms

An important limitation of the K Means algorithm is that the solution found by the algorithm corresponds to a local rather than the global minimum. As a consequence, it is important to run the algorithm a number of times (each time with a different starting configuration) and then select the result that gives the overall lowest sum of within-cluster distances for all documents. A simple check that a solution is robust is to run the algorithm for an increasing number of initial configurations until the result does not change significantly. That said, this procedure does not guarantee a globally optimal solution. To illustrate this point better a wrote a small Matlab code that I have included in the submission. (**mykmeans.m**)

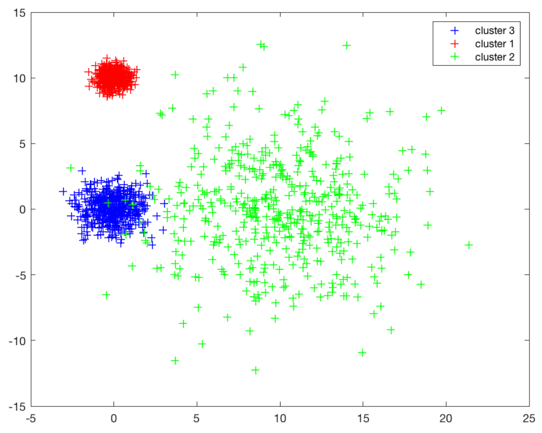


Figure 20. Visualizing the data given in **k_means_data.mat**

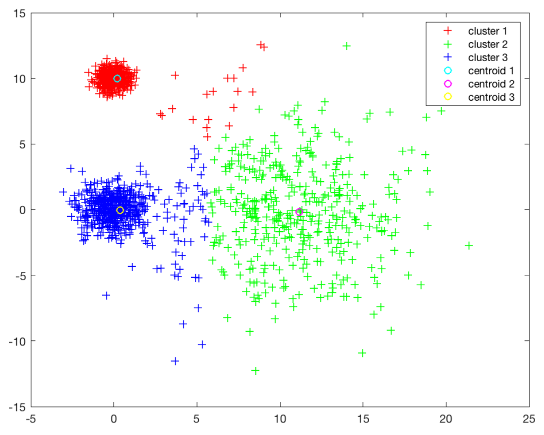


Figure 21. Data clustered by the standard k-means clustering method

As depicted by the above two images, the standard k-means clustering method does a good job of clustering the dataset, with the only exception being the boundary points between two clusters (*rng(3)*). But it is still very **unreliable**. If the initial selection of centroids is not good, we will end up with clusters as shown below. (*rng(27)*). While Hierarchical Clustering has no such limitations.

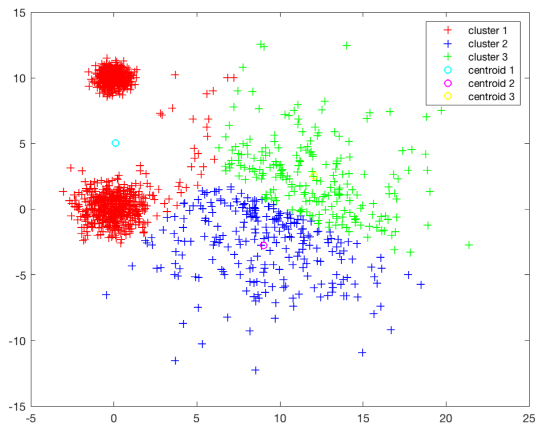


Figure 22. Example of bad clustering

5.2 Optimal number of clusters

The most difficult part of the analysis was finding the number of clusters. The truth is there is no one-size-fits-all answer to this question, but there are some heuristics that can be used.

As I mentioned earlier, the K means algorithm attempts to minimize the sum of the distances between the points in a cluster and the cluster's centroid. The actual quantity that is minimized is the total of the Within-Cluster Sum of Squares (WSS), between each point and the mean. Intuitively one might expect this quantity to be maximum when k=1 and then decreases as k increases, sharply at first and then less sharply as k reaches its optimal value.

Below a plot of summed WSS as a function of k for k=2 all the way to 30. It is evident that summed WSS flattened after k = 25. So the best cluster size for our dataset is 25.

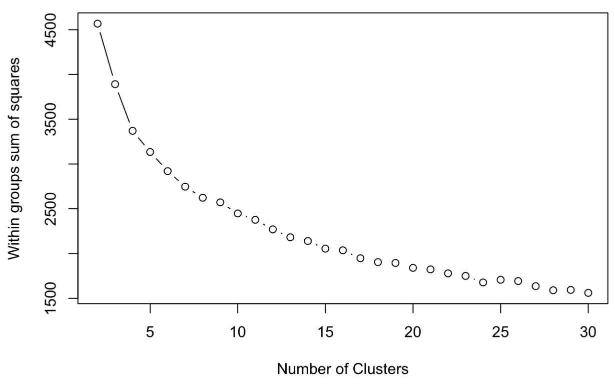


Figure 23. WSS as a function of k

Another worth mentioning point, if you haven't already noticed it. For the number of clusters for both the algorithm that I have used for our Enron email sample data is 7. The motivation behind this was, both Hierarchical Clustering and K Means produced exactly the same clusters with exactly the same documents for this value. I found this after several trials of these algorithms.

References

[1] B. Sarmah, "Text analytics: Mining enron emails." `https://analyticsdefined.com/mining-enron-emails/`.

[2] F. PC, "Enron: The smartest guys in the room." `https://www.youtube.com/watch?v=H2f7FunDuTU`.

[3] Wikipedia, "Document clustering." `https://en.wikipedia.org/wiki/Document_clustering`.

[4] C. U. Press, "Document and query weighting schemes." `https://nlp.stanford.edu/IR-book/html/htmledition/document-and-query-weighting-schemes-1.html`.

[5] Wikipedia, "Cosine similarity." `https://en.wikipedia.org/wiki/Cosine_similarity`.

[6] S. Ali, "Document clustering with r." `https://rpubs.com/saqib/DocumentClustering`.