Big Data: Text Analysis for Competitive Advantage

Manoj Acharya: 808056006

Jared Leach: 800892421

Pratik Parekh: 801076521

Sahil Sood: 801082267

Amal Samirbhai Vaishnav:801063733

Nate Wilson: 801034188

# Introduction

## Problem Space

Big business has been shifting the work of humans to machines since the industrial revolution. Now we are seeing computers encroach on human’s greatest invention, language. Whether it be Google search queries, Amazon chatbots to get you the correct help or even summarizing large legal documents; language is digitizing. Natural language processing is an analysis technique used to distill and correlate bodies of text into a useful summaries and comparisons. This analysis can be distilled into something a computer can understand or simply making it easier for a human to comprehend.

The goal of this project is to discovery the intricacies of language processing and see how we can use the analysis from a business perspective. In this case, we will pose a hypothetical and try to answer it with our algorithms.

## Focus of Analysis

We approached this project from the perspective of a company attempting to utilize natural language processing to summarize and simplify for its various company documents and then specify to what groups/management within the company would use this summarization.

Summarizations are also useful for anyone who needs describers of a large number of works such as in magazine or newspaper divisions, as well as in catalog sites and online libraries.

# Data Preparation

### Removing special characters and stemming

Remove characters that might change the way the word is read. Something as simple as a period could make two words different. Stemming using [Snowball Stemmer](https://snowballstem.org/texts/introduction.html)

from nltk.stem.snowball import SnowballStemmer

### Tokenizing

Each word stem is categorized as separate tokens which allow us to count the number of occurrences in the documents.

nltk.sent\_token and nltk.word\_token

### Removal of stopwords

Common words are removed from the tokens so they do not overwhelm the more unique words that might describe the document. Our list of stopwords are from nltk.

nltk.corpus.stopwords.words('english')

### Removal of Numeric Values

Numeric Values are removed from the tokenized text as they do not play as role or give any insight for our text analysis.The numeric values are removed by placing an isDigit function check for the list of tokens.

# 

# Data Processing

##### Tf/idf algorithm for choosing significant words

from sklearn.feature\_extraction.text import TfidfVectorizer

##### Cosine Similarity for calculating Distance

from sklearn.metrics.pairwise import cosine\_similarity

dist = 1 - cosine\_similarity(tfidf\_matrix)

#### K-means Algorithm for clustering

from sklearn.cluster import KMeans

km.fit(tfidf\_matrix)

#### Dendrogram for hierarchical clustering

from scipy.cluster.hierarchy import ward, dendrogram

#### Multidimensional scaling for 2-dimensional representation

from sklearn.manifold import MDS

# 

# Results

## Visualizations

### Input

We used word clouds to get a clearer picture of the data. This gave us the most common terms in each document and it was easy to scroll through and see which documents may not be useful to us. Here are a range of examples:

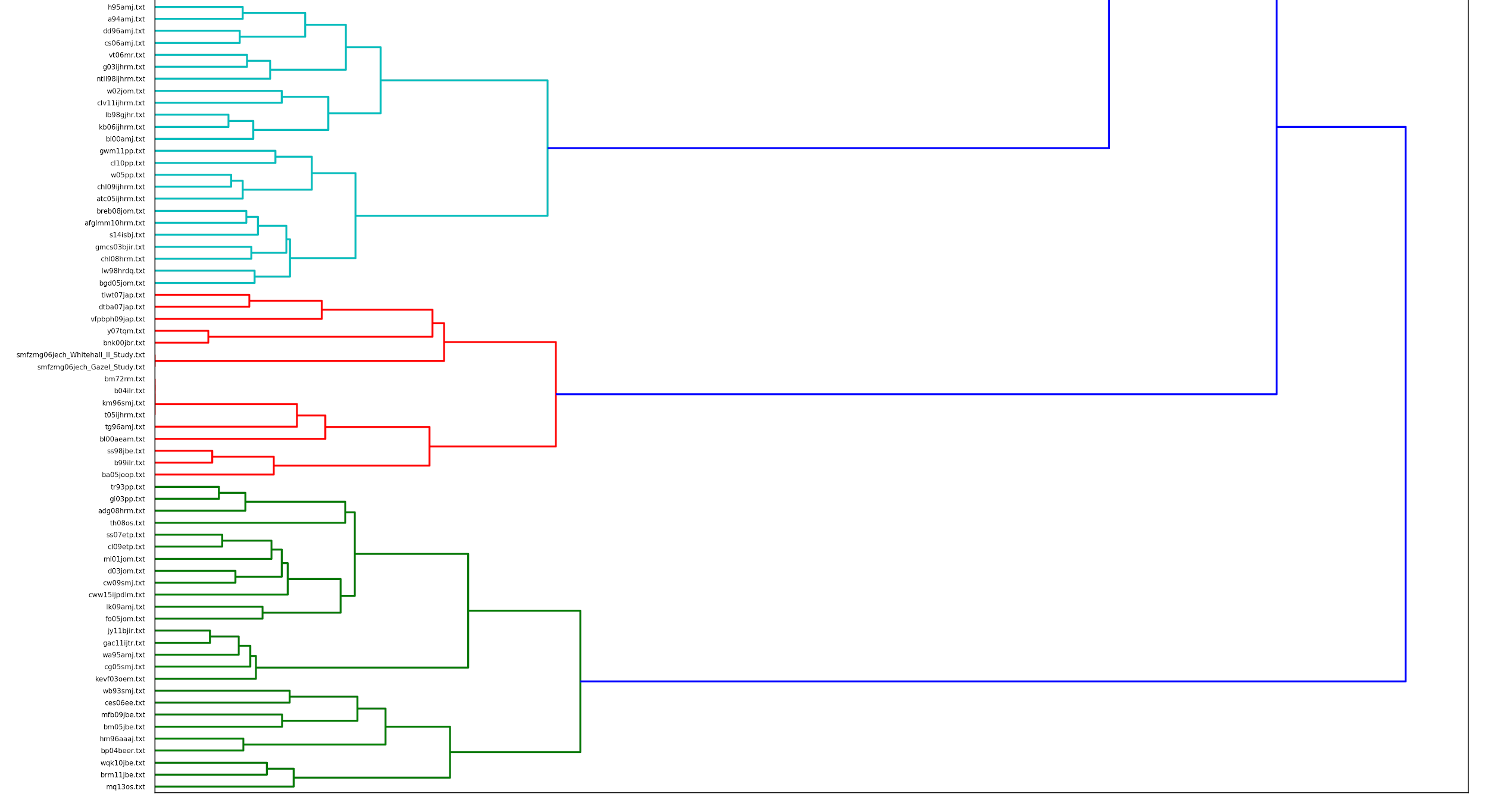
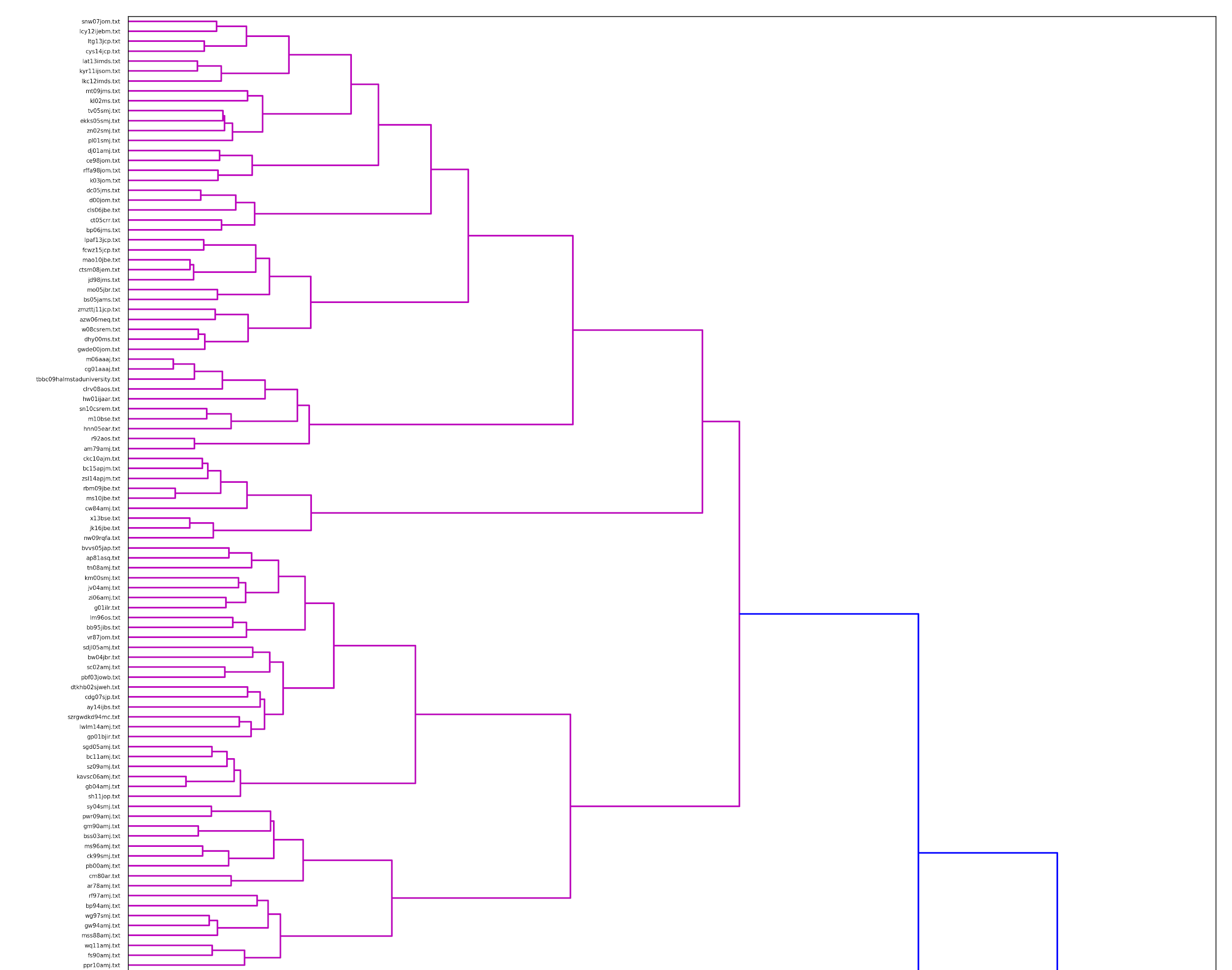
|  |  |
| --- | --- |
|  |  |
| Human resources strategy | Hospital pay structure |
|  |  |
| Some documents had few common term. This one about social responsibility was a bit sparse on repeated words. | Supply chain management. |
|  |  |
| The original file had no content. We tokenized the empty file to have the word EMPTY. Obviously not useable. | This one was odd. It only had “Copyright. All rights reserved” written multiple times. Not usable. |

### Intermediate results

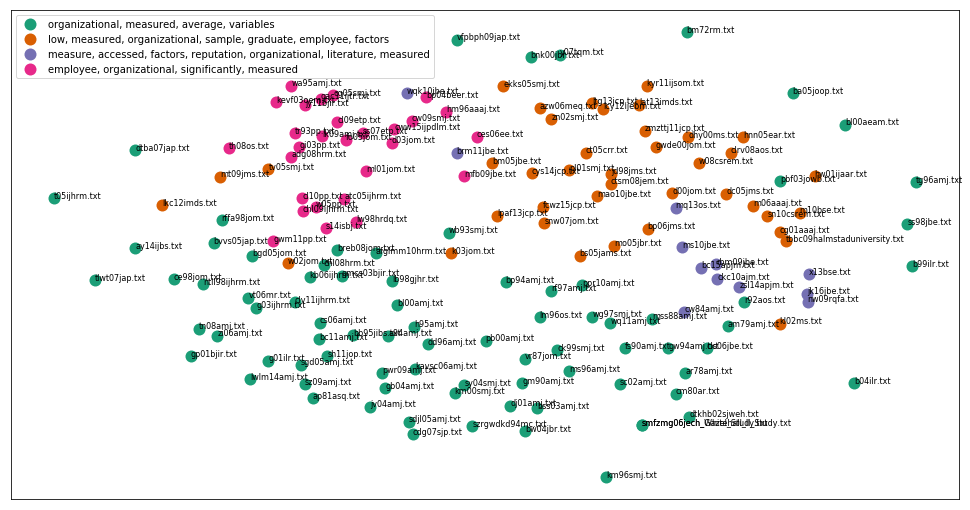
### Final results

We ended up with four clusters:

* Cluster 0 words: positive, statistically, mediating, performance, ing, increase, organization level, available,
* Cluster 1 words: performance, organization-level, job, literature, value,
* Cluster 2 words: performance, englewood, industrial
* Cluster 3 words: high, press, value, use, performance, ing

Dendrogram representation for clustering

Papers plotted through multidimensional scaling(MDS):



Word cloud of all the papers combined:

## 

## 

**Stemmed N-Grams**

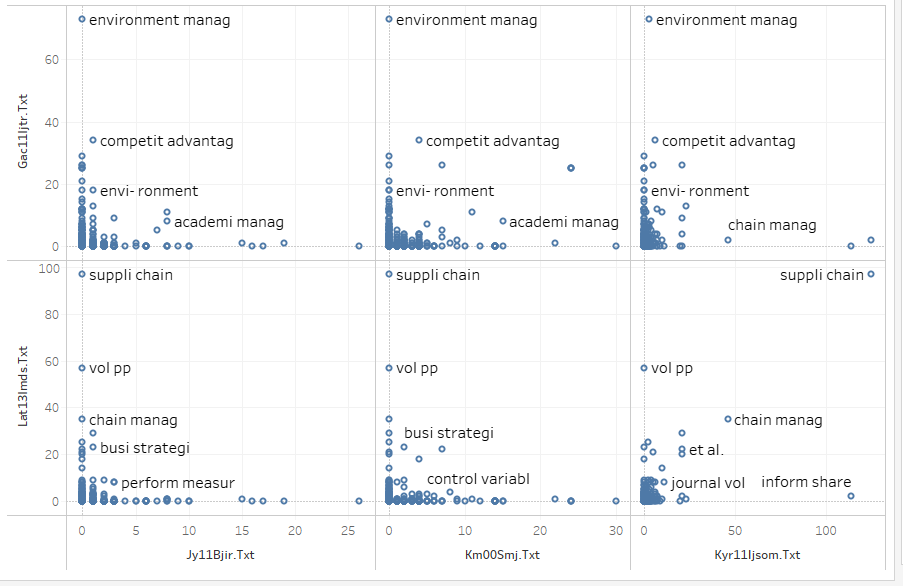
All documents combined:

## 

N-Gram of Kavsc06Amj.Txt:

## 

Document vs Document Comparison:



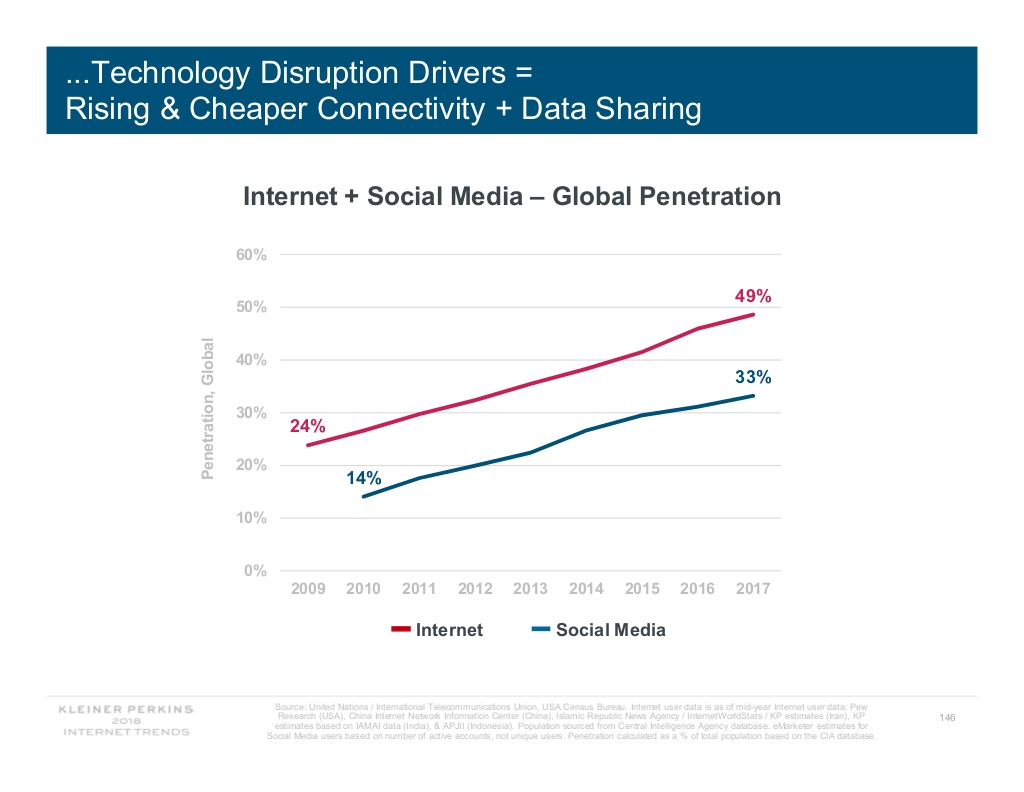
## Interpretation

Through our data exploration, we discovered 4 major groupings within the corpus. The first collection consisted of articles that were denoted by the words "positive, statistically, mediating, performance, ing, increase, organization-level, and available." The second cluster of documents was denoted by the words "performance, organization-level, job, literature, and value." The third cluster was denoted by "performance, ing, engelwood, and industrial," and the fourth cluster was denoted by "high, press, value, use, performance, ing." Moreover, we confirmed the clusters strength by using multidimensional scaling to represent the complex dataset into a simpler form. By labelling the documents by their clusters, we can see physical groupings form on the 2-dimensional graph.

When examining the cluster words, we found it interesting that cluster one, the largest cluster, had nearly twice the number of descriptive words than any of the other clusters, and we were intrigued by how each of the four clusters had the word “performance” describing it. We looked at a handful of documents within each cluster, as well as their word clouds, to determine how well these words correctly summarized the documents. From our cursory findings, the documents all seem to fit the general themes that was set by the k-means algorithm.

# 

# Conclusion



According to the Kleiner-Perkins 2018 report, we are reaching about 50% penetration of internet users. As millions of people get online, they generate an exponentially growing amount of data. To meet the ever-increasing wave of information, businesses need to have tools to distill those articles and texts into their base components. By defining and consolidating these text we can improve efficiency which would reduce time spent searching for relevant information.

On our initial study of the documents provided, we were able to compile four groups that represented the main themes of the documents. This narrowed down the scope of the papers and allowed us the ability to summarize these by association. This project gave us a taste of what large-scale analysis might entail as there are a lot of edge cases that need to be resolved.

If this project were to continue, deeper analysis should be placed on setting different thresholds for the k-means clustering algorithm. Smaller clusters could allow more obscure descriptors to come forth and might do a better job of fleshing out the articles inside. Perhaps describing each article at a different level of abstraction could define the paper in different scopes which could further our understanding of the research space.

# Appendix I

Python Implementation Details

**Import Libraries:**

from \_\_future\_\_ import print\_function

import numpy as np

import pandas as pd

import nltk

import re

import os

import codecs

from sklearn import feature\_extraction

import mpld3

**Sample Stop Words**

stopwords = nltk.corpus.stopwords.words('english')

**Read Files**

xDIR = 'C:/Users/M/Documents/bigdata/files'

titles=[]

corpus=[]

i=0

for f in os.listdir(xDIR):

titles.append(f)

corpus.append(open(os.path.join(xDIR,f), encoding="utf8").read())

i=i+1

**Define Stemmer and Tokenize**

from nltk.stem.snowball import SnowballStemmer

stemmer = SnowballStemmer("english")

def tokenize\_and\_stem(text):

# first tokenize by sentence, then by word

tokens = [word for sent in nltk.sent\_tokenize(text) for word in nltk.word\_tokenize(sent)]

filtered\_tokens = []

# filter out any tokens not containing letters (e.g., numeric tokens, raw punctuation)

for token in tokens:

if re.search('[a-zA-Z]', token):

filtered\_tokens.append(token)

# print(filtered\_tokens[len(filtered\_tokens)-1])

stems = [stemmer.stem(t) for t in filtered\_tokens]

return stems

def tokenize\_only(text):

# first tokenize by sentence, then by word to ensure that punctuation is caught as it's own token

tokens = [word.lower() for sent in nltk.sent\_tokenize(text) for word in nltk.word\_tokenize(sent)]

filtered\_tokens = []

for token in tokens:

if re.search('[a-zA-Z]', token):

filtered\_tokens.append(token)

return filtered\_tokens

def test(corpus):

totalvocab\_stemmed = []

totalvocab\_tokenized = []

allwords\_stemmed = tokenize\_and\_stem(corpus) #for each item in 'corpus', tokenize/stem

totalvocab\_stemmed.extend(allwords\_stemmed) #extend the 'totalvocab\_stemmed' list

allwords\_tokenized = tokenize\_only(corpus)

totalvocab\_tokenized.extend(allwords\_tokenized)

vframe = pd.DataFrame({'words': totalvocab\_tokenized}, index = totalvocab\_stemmed)

# print('there are ' + str(vocab\_frame.shape[0]) + ' items in vocab\_frame')

return vframe

#[totalvocab\_stemmed,totalvocab\_tokenized]

**Create Clusters using TFIDF Algorithm:**

def checkDF(key,dframe):

return key in dframe.index

vocab\_frame=test(corpus[0])

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf\_vectorizer = TfidfVectorizer(max\_df=1.0, max\_features=200000,

min\_df=0.1, stop\_words='english',

use\_idf=True, tokenizer=tokenize\_and\_stem, ngram\_range=(1,3))

tfidf\_matrix = tfidf\_vectorizer.fit\_transform(corpus) #fit the vectorizer to corpus

terms = tfidf\_vectorizer.get\_feature\_names()

terms = list(set(terms))

from sklearn.metrics.pairwise import cosine\_similarity

dist = 1 - cosine\_similarity(tfidf\_matrix)

from sklearn.cluster import KMeans

num\_clusters = 4

km = KMeans(n\_clusters=num\_clusters)

km.fit(tfidf\_matrix)

clusters = km.labels\_.tolist()

from sklearn.externals import joblib

joblib.dump(km, 'doc\_cluster.pkl')

km = joblib.load('doc\_cluster.pkl')

clusters = km.labels\_.tolist()

docs = { 'title': titles, 'synopsis': corpus, 'cluster': clusters }

frame = pd.DataFrame(docs, index = [clusters] , columns = ['title', 'cluster'])

frame['cluster'].value\_counts()

print("Top terms per cluster:")

print()

#sort cluster centers by proximity to centroid

w=[]

order\_centroids = km.cluster\_centers\_.argsort()[:, ::-1]

for i in range(num\_clusters):

print("Cluster %d words:" % i, end='')

for ind in order\_centroids[i, :10]:

wordsToPass=terms[ind].split(' ')

w.append(wordsToPass)

for word in wordsToPass:

if checkDF(word,vocab\_frame)==False:

wordsToPass = [x for x in wordsToPass if x != word]

if len(wordsToPass)>0:

print(' %s' %

vocab\_frame.loc[wordsToPass]

.values

.tolist()[0][0]

.encode('utf-8', 'ignore')

, end=','

)

print()

print()

print("Cluster %d Titles:" % i, end='')

for title in frame.loc[i]['title'].values.tolist():

print(' %s,' % title, end='')

print()

print()

print()

print()

Create the Cluster Diagram (MDS):

import os

import matplotlib.pyplot as plt

import matplotlib as mpl

from sklearn.manifold import MDS

MDS()

# convert two components as we're plotting points in a two-dimensional plane

mds = MDS(n\_components=2, dissimilarity="precomputed", random\_state=1)

pos = mds.fit\_transform(dist) # shape (n\_components, n\_samples)

xs, ys = pos[:, 0], pos[:, 1]

print()

print()

#set up colors per clusters using a dict

cluster\_colors = {0: '#1b9e77', 1: '#d95f02', 2: '#7570b3', 3: '#e7298a', 4: '#66a61e'}

#set up cluster names using a dict

cluster\_names = {0: 'organizational, measured, average, variables',

1: 'low, measured, organizational, sample, graduate, employee, factors',

2: 'measure, accessed, factors, reputation, organizational, literature, measured',

3: 'employee, organizational, significantly, measured',

4: 'Salary, work, wage'}

df = pd.DataFrame(dict(x=xs, y=ys, label=clusters, title=titles))

#group by cluster

groups = df.groupby('label')

# set up plot

fig, ax = plt.subplots(figsize=(17, 9)) # set size

ax.margins(0.05)

#iterate through groups to layer the plot

for name, group in groups:

ax.plot(group.x, group.y, marker='o', linestyle='', ms=12,

label=cluster\_names[name], color=cluster\_colors[name],

mec='none')

ax.set\_aspect('auto')

ax.tick\_params(\

axis= 'x', # changes apply to the x-axis

which='both', # both major and minor ticks are affected

bottom='off', # ticks along the bottom edge are off

top='off', # ticks along the top edge are off

labelbottom='off')

ax.tick\_params(\

axis= 'y', # changes apply to the y-axis

which='both', # both major and minor ticks are affected

left='off', # ticks along the bottom edge are off

top='off', # ticks along the top edge are off

labelleft='off')

ax.legend(numpoints=1) #show legend with only 1 point

#add label in x,y position with the label as the film title

for i in range(len(df)):

ax.text(df.ix[i]['x'], df.ix[i]['y'], df.ix[i]['title'], size=8)

plt.show() #show the plot

TDM:

from sklearn.feature\_extraction.text import CountVectorizer

path="C:/Users/M/Documents/bigdata/tdm.vectorizer.csv"

vectorizer=CountVectorizer(max\_df=1.0, max\_features=200000,min\_df=0.1, stop\_words='english',tokenizer=tokenize\_and\_stem, ngram\_range=(1,3))

XV = vectorizer.fit\_transform(corpus)

tdmArr=XV.toarray()

numpy.savetxt(path,tdmArr , delimiter=",")

readTDM=pd.read\_csv(path, names=vectorizer.get\_feature\_names())

readTDM.insert(0,'DocName',docs['title'])

readTDM['docname']=docs['title']

readTDM.set\_index('docname')

df.rename\_axis('docname')

cols = list(range(1, 20))

readTDM=readTDM.drop(readTDM.columns[cols],axis=1)

readTDM.to\_csv(path, sep=',', encoding='utf-8')

Dendrogram

from scipy.cluster.hierarchy import ward, dendrogram

linkage\_matrix = ward(dist) #using ward clustering distances

fig, ax = plt.subplots(figsize=(15, 20)) # set size

ax = dendrogram(linkage\_matrix, orientation="right", labels=titles);

plt.tick\_params(axis= 'x', which='both', bottom='off',top='off', labelbottom='off')

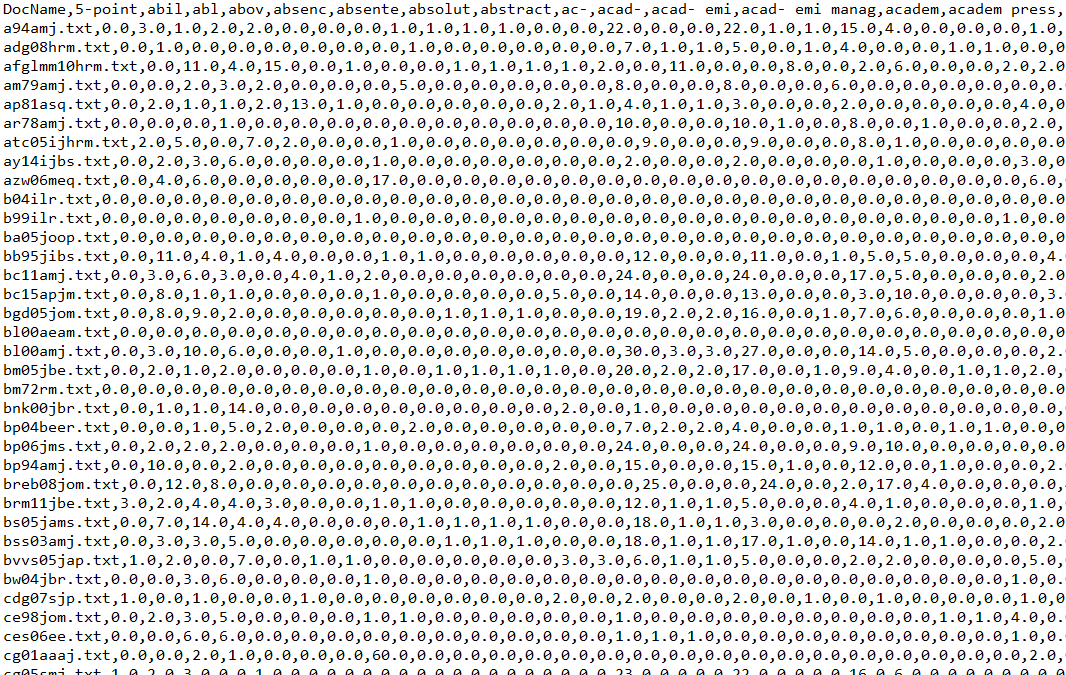
plt.tight\_layout() #show plot with tight layout

plt.savefig('ward\_clusters.png', dpi=300) #save figure as ward\_clusters

# 

# Appendix II

A Sample of the [Term Document Matrix](https://drive.google.com/file/d/1fTgXkMiwRBi88MvyngGg8LhCqSePHDzs/view?usp=sharing)



Tools used :

1. Spyder for Python
2. Sublime Text for regexp find and replace
3. Edit Plus for regexp find and replace
4. Github Desktop and Github.com for source code version control
5. Unix for removing special chars using awk, sed and grep
6. Tableau for n-gram and doc vs doc visualizations