Q1

March 20, 2020

[47]:
```
import pandas as pd
import os, re
import datetime as dt
import re
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
import warnings
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import gensim
from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
import numpy as np
np.random.seed(2018)
import nltk
nltk.download('wordnet')
import warnings
warnings.simplefilter("ignore", DeprecationWarning)

from nltk.stem.snowball import SnowballStemmer

from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt
from sklearn.manifold import MDS
from sklearn.metrics.pairwise import cosine_similarity

from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.metrics import silhouette_score
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/ArshyaSrinivas/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

0.1 Question 1

[29]:
```
#2013 DATA
folder = "/Users/ArshyaSrinivas/Google Drive/Engineering/Junior year/WINTER␣
→2019/IEMS 308/text_analytics/data/2013"
files = []
data = []
for file in os.listdir(folder):
    #get filename without extension
    files.append(file.split('.')[0])
    filepath = os.path.join(folder, file)
    #get file contents
    with open(filepath, encoding="utf8", errors='ignore') as f:
        data.append(f.read())

#convert filename from string to date object to separate by quarter
files = [dt.datetime.strptime(f, '%Y-%m-%d') for f in files]
```

[30]:
```
#create a dataframe obj because it is easy to work on
df = pd.DataFrame(list(zip(files, data)), columns =['date', 'doc'])
```

[31]:
```
#sort df based on date column
df = df.sort_values('date')
```

[5]:
```
df.reset_index(inplace=True)
```

[6]:
```
#clean the data

#remove punctuations
df['doc'] = df['doc'].map(lambda x: re.sub('[,\.!?]', '', x))

#convert to lower case
df['doc'] = df['doc'].map(lambda x: x.lower())


#print first few rows of df
#print(df.head())
```

[7]:
```
#getting the quarter end dates
q1_date_2013 = "2013-03-31"
q2_date_2013 = "2013-06-30"
q3_date_2013 = "2013-09-30"
q4_date_2013 = "2013-12-31"
```

[8]:
```
indicies_2013 = [0]

indicies_2013.append(df.index[df['date'] == q1_date_2013].tolist())
indicies_2013.append(df.index[df['date'] == q2_date_2013].tolist())
indicies_2013.append(df.index[df['date'] == q3_date_2013].tolist())
indicies_2013.append(df.index[df['date'] == q4_date_2013].tolist())
```

[9]:
```
int_indicies_2013 = [0]

for i in range(1,5):
    for j in indicies_2013[i]:
        int_indicies_2013.append(j+1)
```

[10]:
```
q1_df_2013 = df.iloc[0:int_indicies_2013[1],]
q2_df_2013 = df.iloc[int_indicies_2013[1]:int_indicies_2013[2],]
q3_df_2013 = df.iloc[int_indicies_2013[2]:int_indicies_2013[3],]
q4_df_2013 = df.iloc[int_indicies_2013[3]:int_indicies_2013[4],]
```

[11]:
```
stemmer = SnowballStemmer("english")
lemmer=WordNetLemmatizer()

def tokenizer(doc):
    #split doc to words
    words = word_tokenize(doc)
    words = [stemmer.stem(lemmer.lemmatize(word)) for word in words]
    words = [word for word in words if len(word) > 3]
    return words
```

[12]:
```
from sklearn.decomposition import LatentDirichletAllocation as LDA
```

[13]:
```
count_vectorizer = CountVectorizer(stop_words='english', tokenizer=tokenizer)
number_topics = 3
number_words = 5
```

[14]:
```
#Quarter 1 2013
bow_corpus = count_vectorizer.fit_transform(q1_df_2013['doc'])
lda = LDA(n_components=number_topics, n_jobs=-1)
lda.fit(bow_corpus)
# Print the topics found by the LDA model
print("Topics found via LDA:")

words = count_vectorizer.get_feature_names()
for index, topic in enumerate(lda.components_):
    print("\nTopic #%d:" % index)
    print(" ".join([words[i] for i in topic.argsort()[:-number_words - 1:-1]]))
```

```
/Users/ArshyaSrinivas/anaconda3/lib/python3.7/site-
packages/sklearn/feature_extraction/text.py:300: UserWarning: Your stop_words
may be inconsistent with your preprocessing. Tokenizing the stop words generated
tokens ['abov', 'afterward', 'alon', 'alreadi', 'alway', 'anoth', 'anyon',
'anyth', 'anywher', 'becam', 'becaus', 'becom', 'befor', 'besid', 'describ',
'dure', 'elsewher', 'empti', 'everi', 'everyon', 'everyth', 'everywher',
'fifti', 'forti', 'henc', 'hereaft', 'herebi', 'howev', 'hundr', 'inde', 'mani',
'meanwhil', 'moreov', 'nobodi', 'noon', 'noth', 'nowher', 'onli', 'otherwis',
'ourselv', 'perhap', 'pleas', 'sever', 'sinc', 'sincer', 'sixti', 'someon',
'someth', 'sometim', 'somewher', 'themselv', 'thenc', 'thereaft', 'therebi',
```

```
'therefor', 'togeth', 'twelv', 'twenti', 'veri', 'whatev', 'whenc', 'whenev',
'wherea', 'whereaft', 'wherebi', 'wherev', 'yourselv'] not in stop_words.
  'stop_words.' % sorted(inconsistent))

Topics found via LDA:

Topic #0:
year market report percent said

Topic #1:
year market bank percent said

Topic #2:
alcoa gerhartsreit solar iwatch greenberg
```

[15]:
```
#Quarter 2 2013
bow_corpus = count_vectorizer.fit_transform(q2_df_2013['doc'])
lda = LDA(n_components=number_topics, n_jobs=-1)
lda.fit(bow_corpus)
# Print the topics found by the LDA model
print("Topics found via LDA:")

words = count_vectorizer.get_feature_names()
for index, topic in enumerate(lda.components_):
    print("\nTopic #%d:" % index)
    print(" ".join([words[i] for i in topic.argsort()[:-number_words - 1:-1]]))
```

```
Topics found via LDA:

Topic #0:
shaft eurochem snowden sinker soni

Topic #1:
suspect boston tsarnaev polic bomb

Topic #2:
market year said report rate
```

[16]:
```
#Quarter 3 2013
bow_corpus = count_vectorizer.fit_transform(q3_df_2013['doc'])
lda = LDA(n_components=number_topics, n_jobs=-1)
lda.fit(bow_corpus)

# Print the topics found by the LDA model
print("Topics found via LDA:")

words = count_vectorizer.get_feature_names()
```

```
for index, topic in enumerate(lda.components_):
    print("\nTopic #%d:" % index)
    print(" ".join([words[i] for i in topic.argsort()[:-number_words - 1:-1]]))
```

```
Topics found via LDA:

Topic #0:
rate report year market increas

Topic #1:
market year said like time

Topic #2:
trump yahoo board blackberri bank
```

[17]:
```
#Quarter 4 2013
bow_corpus = count_vectorizer.fit_transform(q4_df_2013['doc'])
lda = LDA(n_components=number_topics, n_jobs=-1)
lda.fit(bow_corpus)
# Print the topics found by the LDA model
print("Topics found via LDA:")

words = count_vectorizer.get_feature_names()
for index, topic in enumerate(lda.components_):
    print("\nTopic #%d:" % index)
    print(" ".join([words[i] for i in topic.argsort()[:-number_words - 1:-1]]))
```

```
Topics found via LDA:

Topic #0:
market year said like price

Topic #1:
ahrendt amazoncouk cva/dva burberri gurley

Topic #2:
year market said report like
```

[18]:
```
#2014 DATA

folder = "/Users/ArshyaSrinivas/Google Drive/Engineering/Junior year/WINTER␣
 ↪2019/IEMS 308/text_analytics/data/2014"
files = []
data = []
for file in os.listdir(folder):
    #get filename without extension
    files.append(file.split('.')[0])
```

```
    filepath = os.path.join(folder, file)
    #get file contents
    with open(filepath, encoding="utf8", errors='ignore') as f:
        data.append(f.read())
```

[19]:
```
#convert filename from string to date object to separate by quarter
files = [dt.datetime.strptime(f, '%Y-%m-%d') for f in files]

#create a dataframe obj because it is easy to work on
df = pd.DataFrame(list(zip(files, data)), columns =['date', 'doc'])

#sort df based on date column
df = df.sort_values('date')

df.reset_index(inplace=True)
```

[20]:
```
#clean the data

#remove punctuations
df['doc'] = df['doc'].map(lambda x: re.sub('[,\.!?]', '', x))

#convert to lower case
df['doc'] = df['doc'].map(lambda x: x.lower())
```

[21]:
```
q1_date_2014 = "2014-03-31"
q2_date_2014 = "2014-06-30"
q3_date_2014 = "2014-09-30"
q4_date_2014 = "2014-12-31"

indicies_2014 = [0]

indicies_2014.append(df.index[df['date'] == q1_date_2014].tolist())
indicies_2014.append(df.index[df['date'] == q2_date_2014].tolist())
indicies_2014.append(df.index[df['date'] == q3_date_2014].tolist())
indicies_2014.append(df.index[df['date'] == q4_date_2014].tolist())

int_indicies_2014 = [0]

for i in range(1,5):
    for j in indicies_2014[i]:
        int_indicies_2014.append(j+1)

q1_df_2014 = df.iloc[0:int_indicies_2014[1],]
q2_df_2014 = df.iloc[int_indicies_2014[1]:int_indicies_2014[2],]
q3_df_2014 = df.iloc[int_indicies_2014[2]:int_indicies_2014[3],]
q4_df_2014 = df.iloc[int_indicies_2014[3]:int_indicies_2014[4],]
```

[23]:
```
#Quarter 1 2014
bow_corpus = count_vectorizer.fit_transform(q1_df_2014['doc'])
```

```
lda = LDA(n_components=number_topics, n_jobs=-1)
lda.fit(bow_corpus)
# Print the topics found by the LDA model
print("Topics found via LDA:")

words = count_vectorizer.get_feature_names()
for index, topic in enumerate(lda.components_):
    print("\nTopic #%d:" % index)
    print(" ".join([words[i] for i in topic.argsort()[:-number_words - 1:-1]]))
```

```
Topics found via LDA:

Topic #0:
market year said free appdownload

Topic #1:
koch nonprofit caterpillar dealer disregard

Topic #2:
market year said report bank
```

[24]:

```
#Quarter 2 2014
bow_corpus = count_vectorizer.fit_transform(q2_df_2014['doc'])
lda = LDA(n_components=number_topics, n_jobs=-1)
lda.fit(bow_corpus)
# Print the topics found by the LDA model
print("Topics found via LDA:")

words = count_vectorizer.get_feature_names()
for index, topic in enumerate(lda.components_):
    print("\nTopic #%d:" % index)
    print(" ".join([words[i] for i in topic.argsort()[:-number_words - 1:-1]]))
```

```
Topics found via LDA:

Topic #0:
year said market bank compani

Topic #1:
market year said report compani

Topic #2:
said year market compani free
```

[25]:

```
#Quarter 3 2014
bow_corpus = count_vectorizer.fit_transform(q3_df_2014['doc'])
lda = LDA(n_components=number_topics, n_jobs=-1)
```

```
lda.fit(bow_corpus)
# Print the topics found by the LDA model
print("Topics found via LDA:")

words = count_vectorizer.get_feature_names()
for index, topic in enumerate(lda.components_):
    print("\nTopic #%d:" % index)
    print(" ".join([words[i] for i in topic.argsort()[:-number_words - 1:-1]]))
```

```
Topics found via LDA:

Topic #0:
said year market report compani

Topic #1:
said year compani market report

Topic #2:
said year market compani report
```

[26]:
```
#Quarter 4 2014
bow_corpus = count_vectorizer.fit_transform(q4_df_2014['doc'])
lda = LDA(n_components=number_topics, n_jobs=-1)
lda.fit(bow_corpus)
# Print the topics found by the LDA model
print("Topics found via LDA:")

words = count_vectorizer.get_feature_names()
for index, topic in enumerate(lda.components_):
    print("\nTopic #%d:" % index)
    print(" ".join([words[i] for i in topic.argsort()[:-number_words - 1:-1]]))
```

```
Topics found via LDA:

Topic #0:
year said market report compani

Topic #1:
climax intergener 1235 clan reutersback

Topic #2:
year said market price bank
```

[32]:
```
# running LDA on all the data
folder = "/Users/ArshyaSrinivas/Google Drive/Engineering/Junior year/WINTER␣
 ↪2019/IEMS 308/text_analytics/data/2013"
files = []
```

```
data = []
for file in os.listdir(folder):
    #get filename without extension
    files.append(file.split('.')[0])
    filepath = os.path.join(folder, file)
    #get file contents
    with open(filepath, encoding="utf8", errors='ignore') as f:
        data.append(f.read())

#convert filename from string to date object to separate by quarter
files = [dt.datetime.strptime(f, '%Y-%m-%d') for f in files]

#create a dataframe obj because it is easy to work on
df1 = pd.DataFrame(list(zip(files, data)), columns =['date', 'doc'])

folder = "/Users/ArshyaSrinivas/Google Drive/Engineering/Junior year/WINTER␣
 ↪2019/IEMS 308/text_analytics/data/2014"
files = []
data = []
for file in os.listdir(folder):
    #get filename without extension
    files.append(file.split('.')[0])
    filepath = os.path.join(folder, file)
    #get file contents
    with open(filepath, encoding="utf8", errors='ignore') as f:
        data.append(f.read())

#convert filename from string to date object to separate by quarter
files = [dt.datetime.strptime(f, '%Y-%m-%d') for f in files]

#create a dataframe obj because it is easy to work on
df2 = pd.DataFrame(list(zip(files, data)), columns =['date', 'doc'])

#appending the two databases
df = df1.append(df2)

#sort df based on date column
df = df.sort_values('date')

df.reset_index(inplace=True)
```

[33]:

```
#clean the data

#remove punctuations
df['doc'] = df['doc'].map(lambda x: re.sub('[,\.!?]', '', x))

#convert to lower case
```

```
df['doc'] = df['doc'].map(lambda x: x.lower())
```

[34]:
```
#LDA on all the data

bow_corpus = count_vectorizer.fit_transform(df['doc'])
lda = LDA(n_components=number_topics, n_jobs=-1)
lda.fit(bow_corpus)
# Print the topics found by the LDA model
print("Topics found via LDA:")

words = count_vectorizer.get_feature_names()
for index, topic in enumerate(lda.components_):
    print("\nTopic #%d:" % index)
    print(" ".join([words[i] for i in topic.argsort()[:-number_words - 1:-1]]))
```

```
Topics found via LDA:

Topic #0:
said free appdownload year market

Topic #1:
report said year market price

Topic #2:
year market said bank compani
```

0.2 Question 2

[98]:
```
d = []
for doc in q1_df_2013['doc']:
    d.append(doc)

docs = pd.DataFrame(list(d), columns =['doc'])
```

[99]:
```
#get tf-idf model based feature vectors
tv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True, stop_words='english',␣
↪tokenizer=tokenizer, max_features=500)
tv_matrix = tv.fit_transform(docs['doc'])
tv_matrix = tv_matrix.toarray()

vocab = tv.get_feature_names()
pd.DataFrame(np.round(tv_matrix, 2), columns=vocab)
```

```
/Users/ArshyaSrinivas/anaconda3/lib/python3.7/site-
packages/sklearn/feature_extraction/text.py:300: UserWarning: Your stop_words
may be inconsistent with your preprocessing. Tokenizing the stop words generated
tokens ['abov', 'afterward', 'alon', 'alreadi', 'alway', 'anoth', 'anyon',
'anyth', 'anywher', 'becam', 'becaus', 'becom', 'befor', 'besid', 'describ',
```

```
'dure', 'elsewher', 'empti', 'everi', 'everyon', 'everyth', 'everywher',
'fifti', 'forti', 'henc', 'hereaft', 'herebi', 'howev', 'hundr', 'inde', 'mani',
'meanwhil', 'moreov', 'nobodi', 'noon', 'noth', 'nowher', 'onli', 'otherwis',
'ourselv', 'perhap', 'pleas', 'sever', 'sinc', 'sincer', 'sixti', 'someon',
'someth', 'sometim', 'somewher', 'themselv', 'thenc', 'thereaft', 'therebi',
'therefor', 'togeth', 'twelv', 'twenti', 'veri', 'whatev', 'whenc', 'whenev',
'wherea', 'whereaft', 'wherebi', 'wherev', 'yourselv'] not in stop_words.
  'stop_words.' % sorted(inconsistent))
```

[99]:

	2008	2010	2011	2012	2013	abov	accord	account	ackman	action	...	\
0	0.00	0.00	0.01	0.03	0.01	0.02	0.00	0.00	0.00	0.02	...	
1	0.01	0.02	0.03	0.10	0.08	0.04	0.04	0.02	0.01	0.01	...	
2	0.01	0.01	0.04	0.08	0.09	0.02	0.05	0.04	0.00	0.01	...	
3	0.02	0.02	0.05	0.07	0.11	0.04	0.06	0.02	0.00	0.03	...	
4	0.02	0.02	0.04	0.03	0.11	0.01	0.04	0.04	0.00	0.03	...	
5	0.01	0.00	0.06	0.10	0.06	0.03	0.04	0.02	0.00	0.01	...	
6	0.03	0.02	0.06	0.08	0.09	0.03	0.05	0.03	0.03	0.01	...	
7	0.04	0.01	0.07	0.15	0.09	0.02	0.09	0.02	0.02	0.02	...	
8	0.02	0.02	0.03	0.05	0.09	0.02	0.08	0.04	0.08	0.02	...	
9	0.01	0.01	0.03	0.07	0.06	0.01	0.05	0.02	0.15	0.02	...	
10	0.00	0.02	0.03	0.06	0.03	0.02	0.05	0.01	0.00	0.02	...	
11	0.05	0.01	0.03	0.08	0.19	0.01	0.08	0.01	0.00	0.02	...	
12	0.00	0.01	0.04	0.05	0.03	0.00	0.09	0.01	0.00	0.04	...	
13	0.01	0.05	0.02	0.04	0.06	0.02	0.05	0.02	0.03	0.03	...	
14	0.02	0.01	0.04	0.04	0.05	0.02	0.07	0.03	0.16	0.03	...	
15	0.00	0.01	0.05	0.07	0.07	0.03	0.04	0.02	0.00	0.01	...	
16	0.01	0.01	0.05	0.13	0.07	0.04	0.05	0.02	0.05	0.01	...	
17	0.01	0.01	0.04	0.08	0.05	0.03	0.06	0.02	0.00	0.01	...	
18	0.02	0.03	0.04	0.04	0.04	0.03	0.04	0.10	0.00	0.01	...	
19	0.00	0.00	0.07	0.11	0.09	0.06	0.04	0.00	0.00	0.01	...	
20	0.01	0.01	0.00	0.02	0.08	0.01	0.02	0.01	0.00	0.02	...	
21	0.01	0.01	0.09	0.13	0.09	0.01	0.08	0.04	0.01	0.01	...	
22	0.02	0.00	0.05	0.06	0.03	0.02	0.06	0.04	0.01	0.02	...	
23	0.02	0.01	0.02	0.02	0.06	0.03	0.05	0.01	0.09	0.01	...	
24	0.02	0.01	0.03	0.05	0.04	0.02	0.03	0.03	0.38	0.02	...	
25	0.02	0.02	0.00	0.02	0.02	0.01	0.02	0.03	0.04	0.01	...	
26	0.01	0.03	0.03	0.01	0.03	0.02	0.04	0.05	0.00	0.01	...	
27	0.00	0.01	0.05	0.09	0.07	0.02	0.05	0.01	0.11	0.02	...	
28	0.01	0.01	0.03	0.04	0.08	0.03	0.05	0.01	0.08	0.01	...	
29	0.01	0.01	0.05	0.06	0.11	0.03	0.04	0.03	0.01	0.03	...	
..	...	...	...	...	...	...	...	...	...	...	...	
60	0.08	0.01	0.04	0.04	0.03	0.01	0.03	0.01	0.00	0.00	...	
61	0.01	0.00	0.01	0.02	0.01	0.02	0.07	0.03	0.00	0.03	...	
62	0.01	0.01	0.03	0.08	0.02	0.02	0.08	0.02	0.03	0.01	...	
63	0.01	0.02	0.04	0.04	0.03	0.02	0.07	0.02	0.09	0.01	...	
64	0.00	0.01	0.01	0.05	0.04	0.01	0.04	0.02	0.03	0.00	...	
65	0.00	0.01	0.03	0.07	0.06	0.02	0.09	0.01	0.06	0.01	...	

66	0.01	0.01	0.01	0.04	0.01	0.04	0.07	0.01	0.00	0.04	...
67	0.03	0.03	0.01	0.08	0.09	0.00	0.02	0.02	0.00	0.03	...
68	0.00	0.01	0.01	0.01	0.02	0.01	0.08	0.10	0.00	0.03	...
69	0.01	0.02	0.03	0.02	0.06	0.03	0.08	0.04	0.08	0.02	...
70	0.02	0.03	0.02	0.03	0.02	0.02	0.05	0.02	0.03	0.01	...
71	0.02	0.01	0.01	0.02	0.06	0.01	0.05	0.04	0.03	0.01	...
72	0.02	0.01	0.03	0.06	0.03	0.01	0.07	0.00	0.02	0.01	...
73	0.03	0.01	0.02	0.04	0.05	0.03	0.06	0.01	0.00	0.01	...
74	0.02	0.02	0.02	0.01	0.01	0.00	0.03	0.05	0.00	0.00	...
75	0.00	0.00	0.01	0.02	0.01	0.02	0.02	0.04	0.00	0.01	...
76	0.02	0.01	0.01	0.02	0.04	0.04	0.08	0.06	0.00	0.00	...
77	0.01	0.00	0.00	0.02	0.02	0.01	0.08	0.06	0.00	0.02	...
78	0.01	0.01	0.02	0.01	0.06	0.03	0.07	0.02	0.02	0.02	...
79	0.00	0.03	0.01	0.03	0.06	0.01	0.09	0.01	0.00	0.00	...
80	0.01	0.01	0.03	0.02	0.02	0.02	0.06	0.06	0.00	0.02	...
81	0.01	0.00	0.01	0.02	0.04	0.02	0.06	0.02	0.00	0.01	...
82	0.01	0.01	0.00	0.01	0.02	0.03	0.04	0.04	0.00	0.02	...
83	0.01	0.01	0.03	0.03	0.02	0.01	0.04	0.02	0.00	0.01	...
84	0.01	0.03	0.05	0.04	0.02	0.01	0.03	0.01	0.00	0.00	...
85	0.04	0.01	0.01	0.03	0.02	0.02	0.06	0.03	0.01	0.00	...
86	0.01	0.01	0.02	0.05	0.01	0.03	0.09	0.05	0.00	0.02	...
87	0.02	0.01	0.01	0.01	0.07	0.02	0.05	0.04	0.00	0.02	...
88	0.04	0.01	0.00	0.00	0.01	0.01	0.05	0.03	0.00	0.00	...
89	0.02	0.00	0.01	0.01	0.01	0.04	0.03	0.02	0.00	0.06	...

	worker	world	worri	worth	write	wrote	year	yesterday	yield	york
0	0.00	0.01	0.00	0.00	0.02	0.03	0.21	0.01	0.05	0.00
1	0.02	0.05	0.02	0.01	0.02	0.01	0.23	0.03	0.03	0.03
2	0.03	0.04	0.00	0.02	0.02	0.02	0.21	0.02	0.05	0.04
3	0.01	0.03	0.02	0.01	0.03	0.04	0.29	0.04	0.05	0.03
4	0.00	0.05	0.01	0.01	0.02	0.01	0.19	0.03	0.05	0.08
5	0.00	0.06	0.01	0.01	0.01	0.01	0.31	0.00	0.04	0.01
6	0.00	0.04	0.02	0.03	0.02	0.01	0.19	0.00	0.02	0.02
7	0.01	0.02	0.01	0.03	0.02	0.03	0.24	0.01	0.04	0.03
8	0.01	0.03	0.01	0.02	0.04	0.02	0.19	0.03	0.00	0.04
9	0.01	0.05	0.02	0.02	0.01	0.03	0.19	0.03	0.03	0.08
10	0.01	0.06	0.01	0.02	0.05	0.02	0.20	0.03	0.04	0.02
11	0.00	0.01	0.03	0.02	0.02	0.01	0.21	0.00	0.01	0.08
12	0.00	0.08	0.01	0.00	0.03	0.01	0.18	0.01	0.01	0.01
13	0.00	0.08	0.01	0.01	0.04	0.01	0.17	0.00	0.02	0.01
14	0.02	0.07	0.02	0.02	0.03	0.01	0.21	0.03	0.00	0.04
15	0.03	0.05	0.01	0.01	0.01	0.00	0.21	0.01	0.00	0.04
16	0.00	0.08	0.01	0.02	0.02	0.01	0.27	0.01	0.00	0.01
17	0.00	0.04	0.01	0.02	0.04	0.02	0.19	0.02	0.02	0.03
18	0.12	0.11	0.01	0.05	0.01	0.01	0.23	0.01	0.00	0.01
19	0.00	0.07	0.00	0.02	0.02	0.04	0.25	0.00	0.09	0.01
20	0.02	0.10	0.03	0.02	0.03	0.02	0.19	0.00	0.01	0.01

```
21    0.02  0.10  0.02  0.01  0.02  0.01  0.22       0.00  0.00  0.01
22    0.02  0.09  0.01  0.01  0.04  0.02  0.23       0.01  0.06  0.02
23    0.00  0.07  0.02  0.02  0.03  0.02  0.17       0.02  0.02  0.04
24    0.03  0.05  0.01  0.01  0.01  0.02  0.26       0.01  0.03  0.02
25    0.00  0.13  0.02  0.03  0.04  0.01  0.17       0.00  0.00  0.02
26    0.01  0.13  0.03  0.02  0.04  0.03  0.23       0.01  0.01  0.01
27    0.00  0.13  0.02  0.01  0.05  0.02  0.22       0.01  0.03  0.02
28    0.03  0.06  0.02  0.01  0.08  0.00  0.20       0.03  0.06  0.02
29    0.01  0.06  0.02  0.01  0.03  0.01  0.16       0.03  0.04  0.02
..     ...   ...   ...   ...   ...   ...   ...        ...   ...   ...
60    0.04  0.07  0.02  0.01  0.02  0.01  0.24       0.01  0.04  0.01
61    0.00  0.06  0.01  0.04  0.01  0.03  0.17       0.02  0.04  0.01
62    0.02  0.11  0.01  0.13  0.02  0.01  0.25       0.00  0.03  0.02
63    0.01  0.08  0.01  0.03  0.04  0.02  0.32       0.02  0.04  0.03
64    0.03  0.02  0.01  0.01  0.01  0.01  0.18       0.01  0.00  0.03
65    0.02  0.04  0.02  0.02  0.05  0.02  0.23       0.01  0.02  0.03
66    0.02  0.05  0.00  0.02  0.04  0.02  0.18       0.04  0.00  0.02
67    0.03  0.09  0.01  0.01  0.01  0.01  0.35       0.00  0.01  0.02
68    0.01  0.07  0.00  0.02  0.04  0.02  0.15       0.00  0.00  0.01
69    0.01  0.06  0.03  0.03  0.07  0.04  0.23       0.02  0.02  0.09
70    0.04  0.07  0.03  0.02  0.03  0.02  0.27       0.01  0.04  0.03
71    0.01  0.09  0.01  0.01  0.04  0.03  0.19       0.02  0.01  0.03
72    0.01  0.10  0.01  0.01  0.03  0.02  0.27       0.02  0.01  0.05
73    0.00  0.04  0.00  0.00  0.01  0.03  0.20       0.01  0.02  0.06
74    0.02  0.04  0.01  0.01  0.02  0.00  0.19       0.01  0.00  0.01
75    0.00  0.04  0.03  0.01  0.03  0.02  0.10       0.00  0.00  0.00
76    0.00  0.04  0.02  0.01  0.06  0.02  0.08       0.00  0.02  0.01
77    0.01  0.03  0.02  0.02  0.02  0.02  0.14       0.03  0.01  0.02
78    0.01  0.05  0.01  0.01  0.04  0.01  0.14       0.02  0.02  0.03
79    0.00  0.04  0.03  0.01  0.02  0.02  0.17       0.00  0.00  0.03
80    0.00  0.04  0.01  0.02  0.04  0.02  0.19       0.02  0.06  0.02
81    0.00  0.03  0.01  0.01  0.00  0.00  0.11       0.00  0.03  0.01
82    0.01  0.03  0.00  0.02  0.01  0.01  0.07       0.00  0.00  0.01
83    0.01  0.05  0.02  0.02  0.03  0.00  0.18       0.01  0.00  0.02
84    0.01  0.02  0.01  0.02  0.03  0.01  0.20       0.04  0.04  0.04
85    0.02  0.08  0.01  0.03  0.04  0.01  0.17       0.02  0.01  0.01
86    0.01  0.03  0.02  0.03  0.03  0.01  0.17       0.00  0.02  0.03
87    0.02  0.10  0.01  0.01  0.04  0.03  0.23       0.02  0.02  0.06
88    0.16  0.03  0.02  0.01  0.00  0.00  0.09       0.00  0.00  0.01
89    0.00  0.07  0.01  0.03  0.01  0.02  0.17       0.01  0.02  0.00

[90 rows x 500 columns]
```

[100]:
```
#get lda model based feature vectors
count_vectorizer = CountVectorizer(stop_words='english', tokenizer=tokenizer)

bow_corpus = count_vectorizer.fit_transform(docs['doc'])
```

```python
lda = LDA(n_components=number_topics, n_jobs=-1)
dt_matrix = lda.fit_transform(bow_corpus)

features = pd.DataFrame(dt_matrix)
features
```

```
/Users/ArshyaSrinivas/anaconda3/lib/python3.7/site-
packages/sklearn/feature_extraction/text.py:300: UserWarning: Your stop_words
may be inconsistent with your preprocessing. Tokenizing the stop words generated
tokens ['abov', 'afterward', 'alon', 'alreadi', 'alway', 'anoth', 'anyon',
'anyth', 'anywher', 'becam', 'becaus', 'becom', 'befor', 'besid', 'describ',
'dure', 'elsewher', 'empti', 'everi', 'everyon', 'everyth', 'everywher',
'fifti', 'forti', 'henc', 'hereaft', 'herebi', 'howev', 'hundr', 'inde', 'mani',
'meanwhil', 'moreov', 'nobodi', 'noon', 'noth', 'nowher', 'onli', 'otherwis',
'ourselv', 'perhap', 'pleas', 'sever', 'sinc', 'sincer', 'sixti', 'someon',
'someth', 'sometim', 'somewher', 'themselv', 'thenc', 'thereaft', 'therebi',
'therefor', 'togeth', 'twelv', 'twenti', 'veri', 'whatev', 'whenc', 'whenev',
'wherea', 'whereaft', 'wherebi', 'wherev', 'yourselv'] not in stop_words.
  'stop_words.' % sorted(inconsistent))
```

[100]:

	0	1	2
0	0.035211	0.964697	0.000092
1	0.010495	0.989478	0.000027
2	0.999942	0.000030	0.000028
3	0.343850	0.656109	0.000040
4	0.989473	0.010451	0.000076
5	0.129614	0.870289	0.000097
6	0.443494	0.096155	0.460350
7	0.481766	0.018718	0.499516
8	0.276417	0.346313	0.377270
9	0.054790	0.945181	0.000029
10	0.672957	0.003083	0.323960
11	0.299286	0.700608	0.000106
12	0.174587	0.375000	0.450412
13	0.004453	0.995515	0.000032
14	0.998267	0.001704	0.000029
15	0.999967	0.000017	0.000016
16	0.218738	0.781237	0.000025
17	0.852997	0.146957	0.000046
18	0.999813	0.000095	0.000092
19	0.129093	0.870756	0.000152
20	0.000072	0.999857	0.000071
21	0.998478	0.001496	0.000025
22	0.000601	0.999376	0.000023
23	0.034927	0.965044	0.000029
24	0.134407	0.865565	0.000027

```
25  0.920152  0.079747  0.000101
26  0.122787  0.877131  0.000082
27  0.997698  0.002272  0.000031
28  0.000039  0.999923  0.000037
29  0.831530  0.003522  0.164948
..       ...       ...       ...
60  0.998198  0.001721  0.000081
61  0.112395  0.420130  0.467475
62  0.990778  0.009179  0.000043
63  0.013658  0.986304  0.000038
64  0.999961  0.000020  0.000019
65  0.997392  0.002564  0.000044
66  0.999874  0.000065  0.000061
67  0.019536  0.980343  0.000121
68  0.007669  0.992212  0.000119
69  0.068660  0.931295  0.000045
70  0.973259  0.026710  0.000031
71  0.003427  0.996531  0.000042
72  0.999284  0.000686  0.000029
73  0.696277  0.000726  0.302998
74  0.000088  0.999829  0.000083
75  0.000061  0.999880  0.000059
76  0.000042  0.999918  0.000039
77  0.000056  0.999892  0.000052
78  0.078277  0.921682  0.000041
79  0.001018  0.998931  0.000052
80  0.021949  0.667968  0.310083
81  0.000078  0.999850  0.000073
82  0.000077  0.999850  0.000073
83  0.000046  0.999911  0.000043
84  0.010804  0.989146  0.000050
85  0.000180  0.999774  0.000046
86  0.137440  0.634339  0.228221
87  0.013078  0.986857  0.000065
88  0.000070  0.999863  0.000067
89  0.000059  0.663789  0.336152

[90 rows x 3 columns]
```

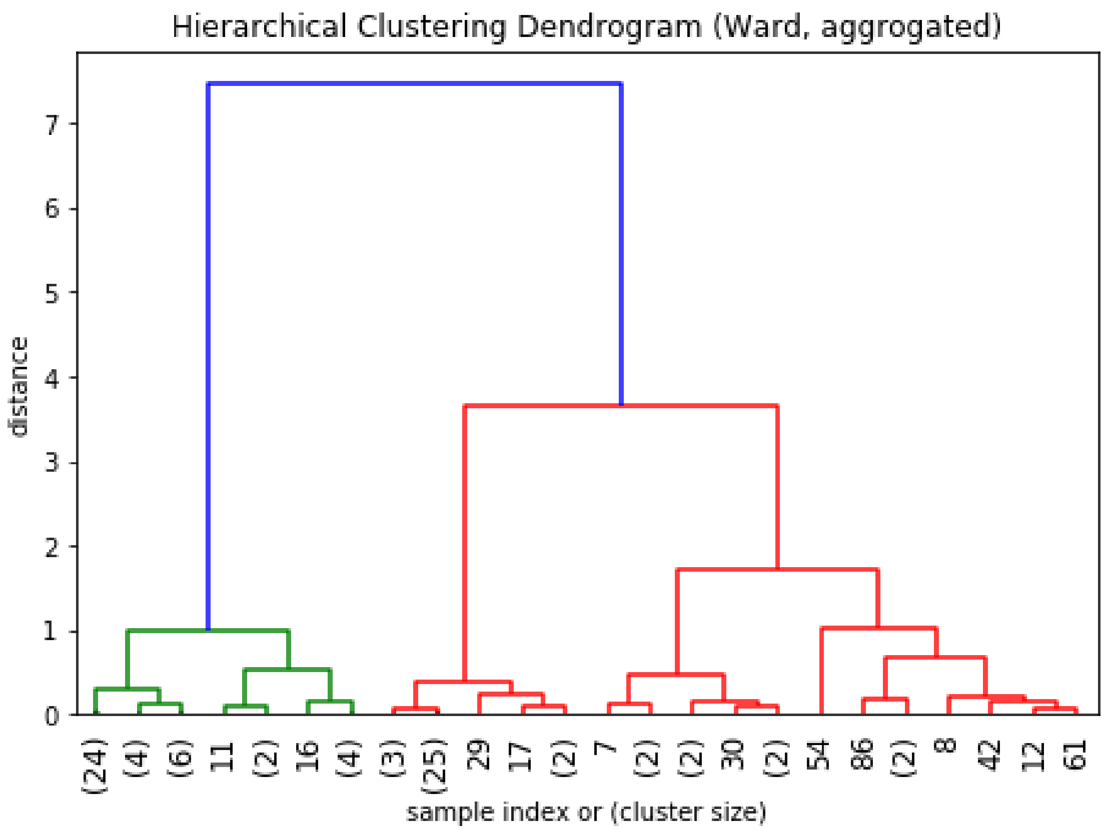
[128]:
```
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
import scipy.cluster.hierarchy as sch
```

[174]:
```
clustering = AgglomerativeClustering(linkage="ward").fit(features)
```

[175]:
```
linkage_matrix = linkage(features, 'ward')
```

[176]:
```
figure = plt.figure(figsize=(7.5, 5))
dendrogram(
```

```
    linkage_matrix,
    truncate_mode='lastp',  # show only the last p merged clusters
    p=24,  # show only the last p merged clusters
    leaf_rotation=90.,
    leaf_font_size=12.,
    show_contracted=True,  # to get a distribution impression in truncated␣
 ↪branches
)
plt.title('Hierarchical Clustering Dendrogram (Ward, aggrogated)')
plt.xlabel('sample index or (cluster size)')
plt.ylabel('distance')
plt.show()
```



[189]:
```
model = AgglomerativeClustering(n_clusters=2, affinity='euclidean',␣
 ↪linkage='ward')
model.fit(features)
labels = model.labels_
```

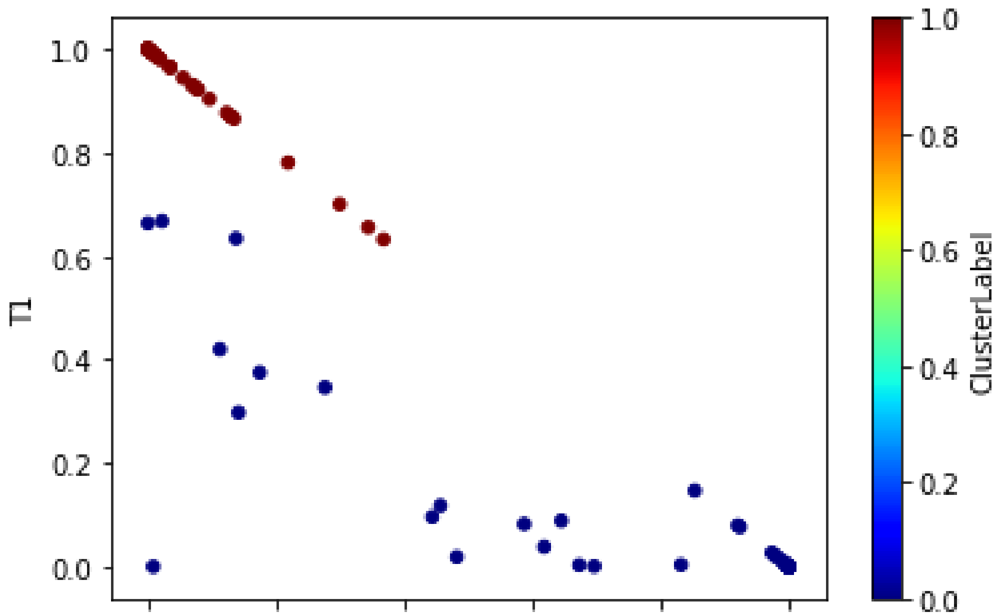
[193]:
```
label = pd.DataFrame(labels, columns = ["ClusterLabel"])
```

[195]:
```
features["ClusterLabel"] = label
```

[224]:
```
features.rename(columns={features.columns[0]: "T0" }, inplace = True)
features.rename(columns={features.columns[1]: "T1" }, inplace = True)
features.rename(columns={features.columns[2]: "T2" }, inplace = True)
```

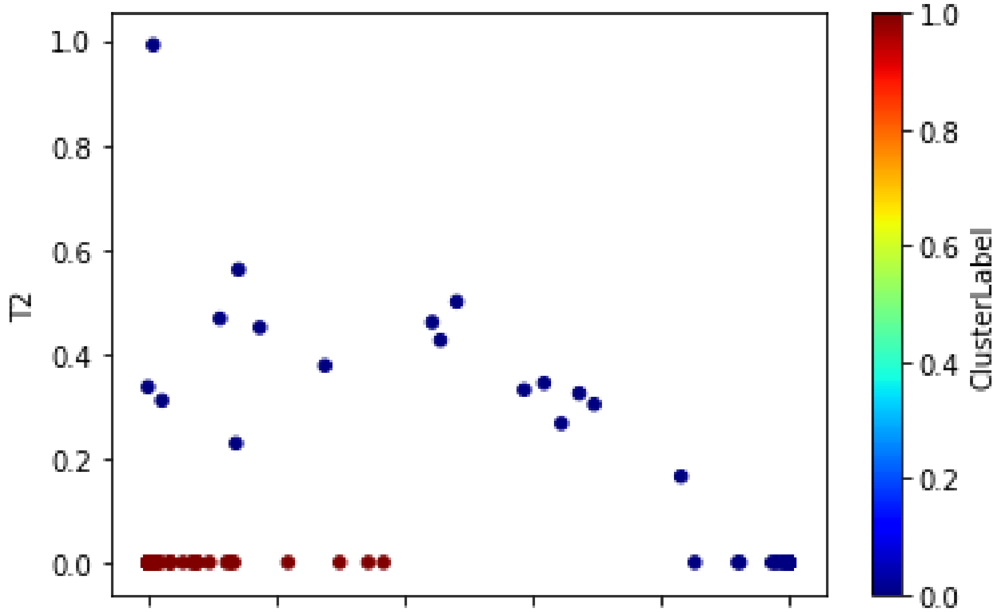
[228]:
```
features.plot.scatter('T0', 'T1', c = "ClusterLabel", cmap='jet')
```

[228]: <matplotlib.axes._subplots.AxesSubplot at 0x1a46f084e0>



[229]:
```
features.plot.scatter('T0', 'T2', c = "ClusterLabel", cmap='jet')
```

[229]: <matplotlib.axes._subplots.AxesSubplot at 0x1a4715bb70>



[230]:
```
features.plot.scatter('T1', 'T2', c = "ClusterLabel", cmap='jet')
```

[230]: <matplotlib.axes._subplots.AxesSubplot at 0x1a47238a20>