CS25025_HW2_P3

April 17, 2018

CS25025_HW2_P4

```
In [420]: import numpy as np
          import pandas as pd
          import matplotlib.pyplot as plt
          import sklearn
          import pickle
          import pyspark
          import nltk
          nltk.download('punkt')
          import itertools
          import string
          import os
          import re
          import math
          from scipy.spatial import distance_matrix
```

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

```
In [421]: f = open("speeches.pkl",'rb')
          speeches = pickle.load(f)
```

```
In [422]: speeches = pd.DataFrame(speeches,columns = ['president','words','year'])
```

(a) Compute TF-TDF for Each SOU Address

```
In [423]: ## clean the data
          def clean_and_split(s):
                      # encode to UTF-8, convert to lowercase and translate all hyphens and
                      # punctuation to whitespace
                      s = s.lower().replace('-',' ').translate(string.punctuation)
                      # replace \r\n
                      s = re.sub('(\r\n)+',' ',s)
                      # replace whitespace substrings with one whitespace and remove
                      # leading/trailing whitespaces
                      s = re.sub(' +',' ',s.strip())
                      return s.split(' ')
```

```
In [424]: speeches['cleaned'] = speeches['words'].apply(clean_and_split)

In [425]: ## decide vocabulary
          voc_pool = list(itertools.chain.from_iterable(speeches['cleaned']))
          voc_counts = pd.value_counts(voc_pool)
          voc_counts = pd.DataFrame(voc_counts)
          voc_counts = voc_counts.loc[voc_counts.loc[:,0] > 50,:]
          voc_counts = voc_counts.iloc[20:,:]
          voc = voc_counts.index

In [426]: ## begin to compute "TF-IDF"
          from collections import Counter
          word_counter = [Counter(x) for x in speeches['cleaned']]
          ## speech["TF-IDF"] here is now only the n_i(d)
          speeches["TF-IDF"] = [pd.DataFrame([y[x] for x in voc],index = voc)\
                                for y in word_counter]

In [427]: ## compute the global weight parameter
          D = len(speeches["TF-IDF"])
          d = [sum([int(x.loc[y].values[0]>0) for x in speeches["TF-IDF"]]) for y in voc]

In [428]: global_weight = [math.log(D/x) for x in d]

In [429]: ## now obtain the "TF-IDF" desired
          speeches["TF-IDF"] = [[x.values[:,0]*global_weight] for x in speeches["TF-IDF"]]
          #speeches["TF-IDF"][d][0][i]: dth row (speech) and i th word in voc; 0 is fixed
```

(b)measuring distance between documents 50 most similar pairs of SOUs given by different Presidents. 50 most similar pairs of SOUs given by the same President. 25 most similar pairs of Presidents, averaging the cosine similarity over all pairs of their SOUs.

```
In [430]: ## extract information of speeches["TF-IDF"] in we, meaning, the weight information
          we = [list(x[0]) for x in speeches["TF-IDF"]]
          #speeches["weights"] = [list(x[0]) for x in speeches["TF-IDF"]]
          #speech_weights = speeches["weights"].copy()

In [103]: we = np.array(we)

In [107]: ## build a distance matrix with metric cosine distance
          cos_dis = sklearn.metrics.pairwise.pairwise_distances(we,we,metric='cosine')

In [108]: # Obtain cosine similarity matrix (Cosine distance is defined as 1.0 minus the cosin
          cos_sim = 1 - cos_dis

In [109]: # take the upper triangle, including the diagonal elements of the similarity matrix
          cos_sim = np.triu(cos_sim,k=1)

In [113]: ## indicator matrix for the same president: 1 for the same president; 0 for differen
          idc_same_pre = [[int(speeches.loc[x,'president']==speeches.loc[y,'president']) for y
```

In [114]:
```
# take the upper triangle, including the diagonal elements of the indicator matrix
idc_same_pre = np.array(idc_same_pre)
idc_same_pre = np.triu(idc_same_pre,k=1)
```

In [115]:
```
## same_pre_sim contains only the sim between speeches of the same president, with O
## diff_pre_sim contains only the sim between speeches of the different president, w
same_pre_sim = np.multiply(idc_same_pre,cos_sim)
diff_pre_sim = cos_sim-same_pre_sim
```

In [116]:
```
## sort the matrix and find the most similar pairs by different presidents
ind_diff_pre = np.unravel_index(np.argsort(-1*diff_pre_sim, axis=None), diff_pre_sim
# sort on (-1*diff_pre_simargsort) because this function results in increasing order
```

In [119]:
```
diff_pre_top50 = [(speeches.loc[ind_diff_pre[0][i],'year'],\
                   speeches.loc[ind_diff_pre[1][i],'year'])for i in range(50)]
```

In [120]:
```
## this is the top 50 speeches between different presidents
diff_pre_top50
```

Out[120]:
```
[('1864', '1882'),
 ('1883', '1864'),
 ('1882', '1872'),
 ('1883', '1872'),
 ('1864', '1872'),
 ('1881', '1872'),
 ('1864', '1881'),
 ('1862', '1882'),
 ('1883', '1862'),
 ('1862', '1872'),
 ('1884', '1872'),
 ('1864', '1884'),
 ('1862', '1881'),
 ('1910', '1882'),
 ('1910', '1872'),
 ('1883', '1910'),
 ('1910', '1881'),
 ('1880', '1881'),
 ('1864', '1910'),
 ('1862', '1884'),
 ('1880', '1872'),
 ('1880', '1882'),
 ('1883', '1880'),
 ('1910', '1884'),
 ('1880', '1884'),
 ('1864', '1880'),
 ('1880', '1910'),
 ('1862', '1910'),
 ('1862', '1880'),
 ('1874', '1881'),
```

```
 ('1874', '1884'),
 ('1874', '1880'),
 ('1874', '1910'),
 ('1883', '1874'),
 ('1874', '1882'),
 ('1862', '1874'),
 ('1874', '1864'),
 ('1880', '1896'),
 ('1896', '1881'),
 ('1910', '1896'),
 ('1896', '1884'),
 ('1896', '1872'),
 ('1880', '1888'),
 ('1896', '1882'),
 ('1862', '1896'),
 ('1883', '1896'),
 ('1864', '1896'),
 ('1881', '1888'),
 ('1910', '1888'),
 ('1884', '1888')]
```

In [121]:
```
## sort the matrix and find the most similar pairs by the same presidents
ind_same_pre = np.unravel_index(np.argsort(-1*same_pre_sim, axis=None), same_pre_sim
# sort on (-1*same_pre_simargsort) because this function results in increasing order
```

In [122]:
```
same_pre_top50 = [(speeches.loc[ind_same_pre[0][i],'year'],\
                   speeches.loc[ind_same_pre[1][i],'year'])for i in range(50)]
```

In [123]:
```
## this is the top 50 speeches by the same presidents
same_pre_top50
```

Out[123]:
```
[('1883', '1882'),
 ('1882', '1881'),
 ('1883', '1881'),
 ('1862', '1864'),
 ('1883', '1884'),
 ('1884', '1882'),
 ('1884', '1881'),
 ('1874', '1872'),
 ('2010', '2011'),
 ('2010', '2012'),
 ('2011', '2012'),
 ('1877', '1880'),
 ('1896', '1888'),
 ('2013', '2012'),
 ('2007', '2008'),
 ('1998', '1999'),
 ('1966', '1967'),
 ('2010', '2009'),
```

```
 ('1998', '2000'),
 ('1998', '1997'),
 ('1995', '1994'),
 ('2009', '2012'),
 ('1905', '1907'),
 ('2009', '2011'),
 ('1999', '2000'),
 ('1908', '1907'),
 ('1900', '1899'),
 ('1999', '1997'),
 ('1899', '1898'),
 ('1955', '1956'),
 ('2006', '2007'),
 ('2008', '2005'),
 ('1905', '1904'),
 ('1845', '1846'),
 ('1907', '1906'),
 ('1894', '1893'),
 ('1905', '1901'),
 ('1905', '1906'),
 ('1911', '1912'),
 ('1892', '1891'),
 ('1886', '1885'),
 ('1835', '1834'),
 ('2006', '2008'),
 ('2003', '2007'),
 ('1909', '1912'),
 ('2007', '2004'),
 ('1994', '1993'),
 ('1956', '1954'),
 ('2007', '2005'),
 ('1997', '2000')]
```

In [347]:
```
## now find most similar pairs of presidents
```

In [124]:
```
## cos_sim_df is now a full symmetric matrix, with diagnol 0
cos_sim_df = pd.DataFrame(cos_sim+np.transpose(cos_sim),index = speeches['president']
```

In [125]:
```
president = np.unique(speeches['president'])
```

In [126]:
```
## it might happen that the matrix is reduced to a float after second first .mean()
def safe_mean(x):
    if isinstance(x,float):
        return x
    else:
        return x.mean()
```

In [127]:
```
### pre_sim is a similarity matrix of presidents
#fp = [safe_mean(cos_sim_df.loc[president[0], x].mean()) for x in president]
```

```
pre_sim = np.array([[np.array(safe_mean(cos_sim_df.loc[y, x].mean())) for x in presi
# pre_sim[i][j] the ith president with his jth friend
```

In [128]:
```
pre_sim = np.triu(pre_sim,k=1)
pre_pair_ind = np.unravel_index(np.argsort(-1*pre_sim, axis=None), pre_sim.shape)
```

In [129]:
```
pre_top25 = [(president[pre_pair_ind[0][i]],\
                        president[pre_pair_ind[1][i]])for i in range(25)]
```

In [130]:
```
## this is top 25 pair of presidents
pre_top25
```

Out[130]:
```
[('Abraham Lincoln', 'Chester A. Arthur'),
 ('Barack Obama', 'William J. Clinton'),
 ('Millard Fillmore', 'Zachary Taylor'),
 ('Barack Obama', 'George Bush'),
 ('George Bush', 'Ronald Reagan'),
 ('George Bush', 'William J. Clinton'),
 ('Chester A. Arthur', 'Rutherford B. Hayes'),
 ('Ronald Reagan', 'William J. Clinton'),
 ('Benjamin Harrison', 'Grover Cleveland'),
 ('Andrew Jackson', 'Martin Van Buren'),
 ('Barack Obama', 'Ronald Reagan'),
 ('James K. Polk', 'Millard Fillmore'),
 ('Chester A. Arthur', 'William Howard Taft'),
 ('Franklin Pierce', 'Millard Fillmore'),
 ('Benjamin Harrison', 'William McKinley'),
 ('Franklin Pierce', 'James Buchanan'),
 ('James K. Polk', 'Zachary Taylor'),
 ('Dwight D. Eisenhower', 'John F. Kennedy'),
 ('James Buchanan', 'Zachary Taylor'),
 ('Dwight D. Eisenhower', 'Harry S Truman'),
 ('John Tyler', 'Martin Van Buren'),
 ('James K. Polk', 'John Tyler'),
 ('Franklin Pierce', 'Zachary Taylor'),
 ('Rutherford B. Hayes', 'William Howard Taft'),
 ('Grover Cleveland', 'Rutherford B. Hayes')]
```

Finding, first,presidents from the same party tends to be similar in style; second, president from the same period of time tend to emulate each other in speech.

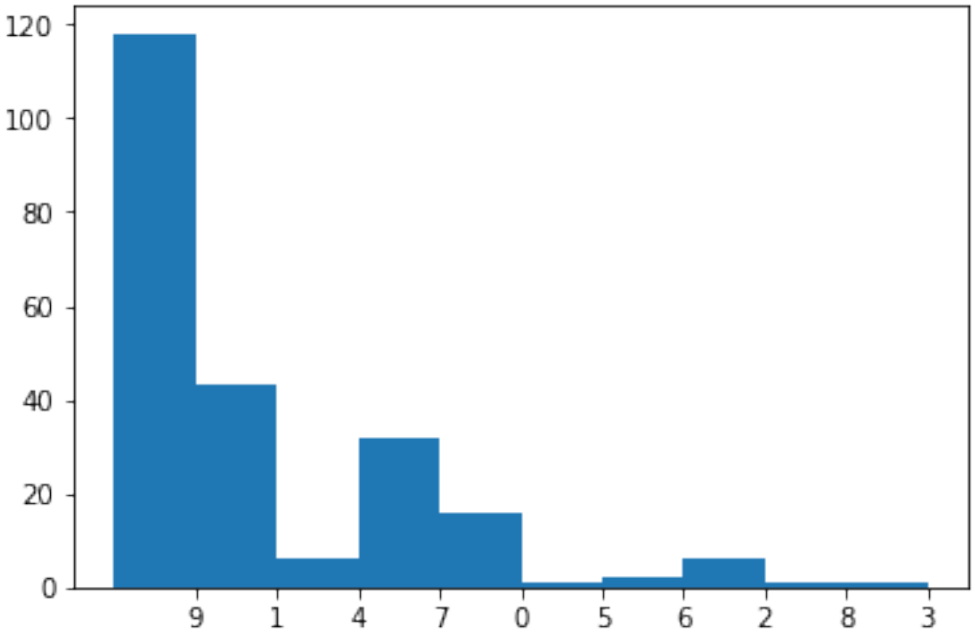
Need to improve: First, the length of the document should be considered when constructing the distance matrix, as the length will influence the the occurence of certain words.

Second, people use different words in different times, the words of of Abraham Lincoln might be very different from Barack Obama. Therefore,we need to add a coefficient of time distance to represent the possible word differences due to the passing pf time.

(c) clustering

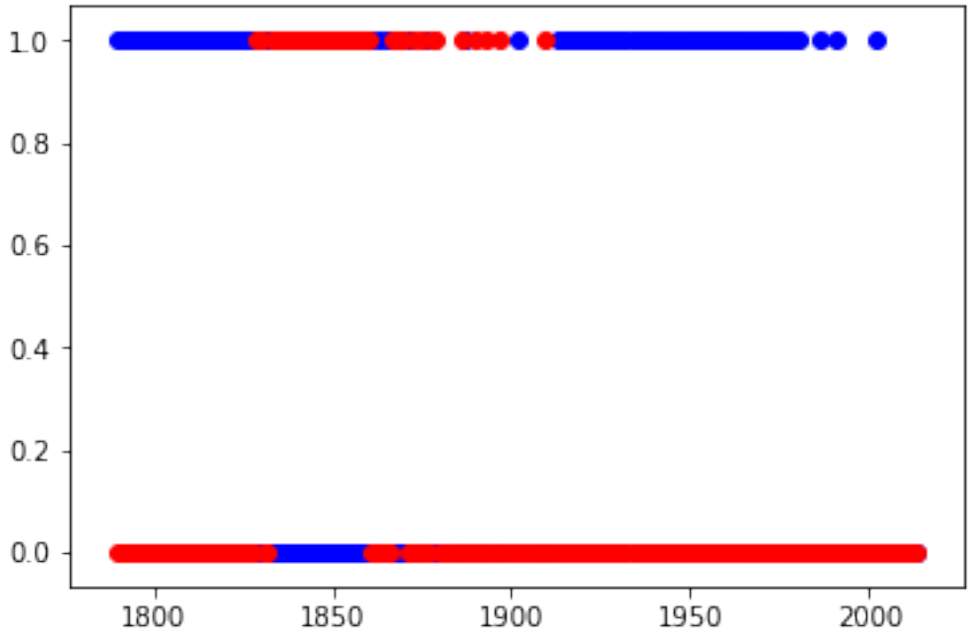
In [133]:
```
from sklearn.cluster import KMeans
```

```
In [143]: #tf_idf_mat = cos_sim.copy()
          #tf_idf_mat = [list(x) for x in cos_sim]
          we.shape

Out[143]: (226, 3279)

In [145]: model = KMeans(n_clusters=10)
          #sou_clust = model.fit(tf_idf_mat)
          sou_clust = model.fit(we)
          # we has the weight information (the vector representation of every speech)

In [146]: #y_means = sou_clust.predict(tf_idf_mat)
          y_means = sou_clust.predict(we)

In [393]: centers = sou_clust.cluster_centers_
          #plt.scatter(tf_idf_mat,c=y_means, cmap='viridis')
          #plt.scatter(centers[:,0], centers[:,1], c = '0.5', s= 200, alpha = 0.8)
          #centers

In [148]: centers.shape

Out[148]: (10, 3279)

In [163]: from collections import Counter
          cluster_num_10 = Counter(y_means)

In [164]: ## find the cluster with the most speeches
          most_common_10 = cluster_num.most_common(2)
          most_common_10

Out[164]: [(9, 118), (1, 43)]

In [361]: ## blow show the distribution of the speeches among various clusters
          from collections import Counter
          labels, values = zip(*Counter(y_means).items())

          indexes = np.arange(len(labels))
          width = 1

          plt.bar(indexes, values, width)
          plt.xticks(indexes + width * 0.5, labels)
          plt.show()
```



In [368]:
```
## from the graph above we can see cluster 9 and 1 contains most of the speeches, th
years = [int(x) for x in speeches['year']]
cluster_year = [[int(x == y) for x in y_means] for y in range(10)]
plt.scatter(years,cluster_year[9],c = 'b')
plt.scatter(years,cluster_year[1],c = 'r')
```

Out[368]: <matplotlib.collections.PathCollection at 0x1a22a1afd0>

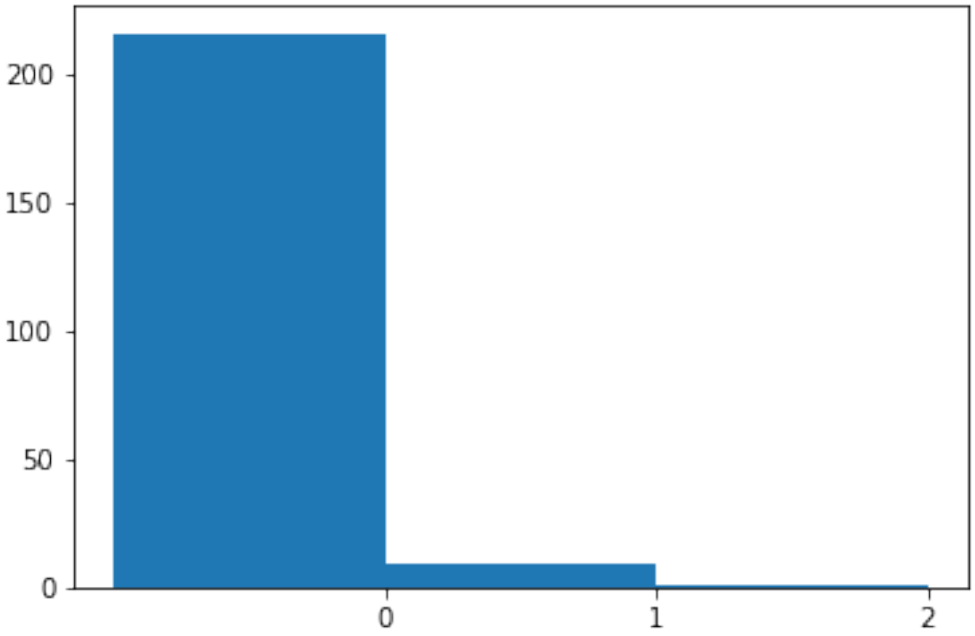


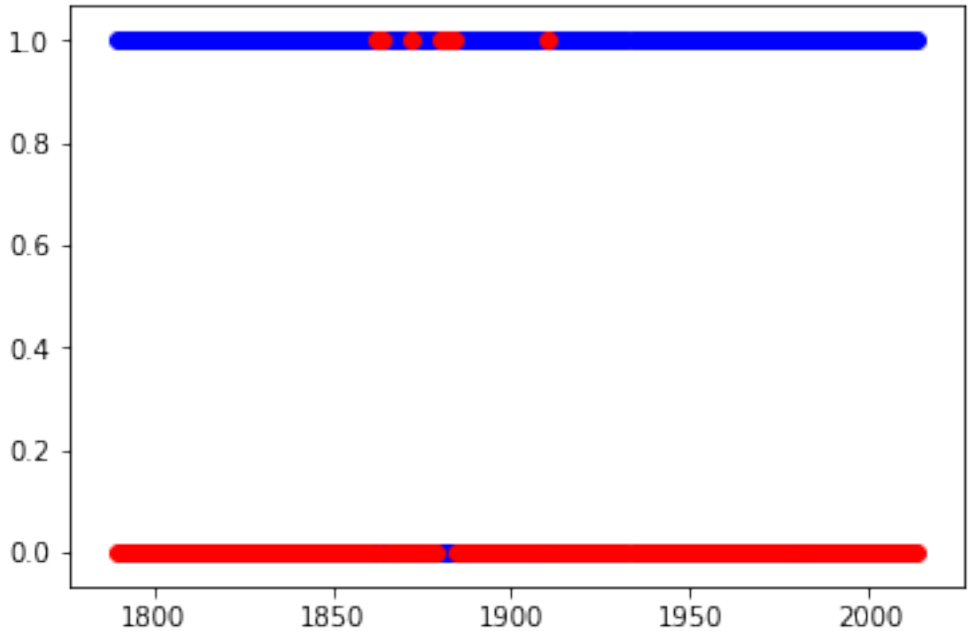
In [ ]:
```
## from the graph above, we can see alternating occurrence of the two types
```

In [375]:
```
#indices = [[i for i, x in enumerate(cluster_year[j]) if x == 1] for j in range(10)]
```

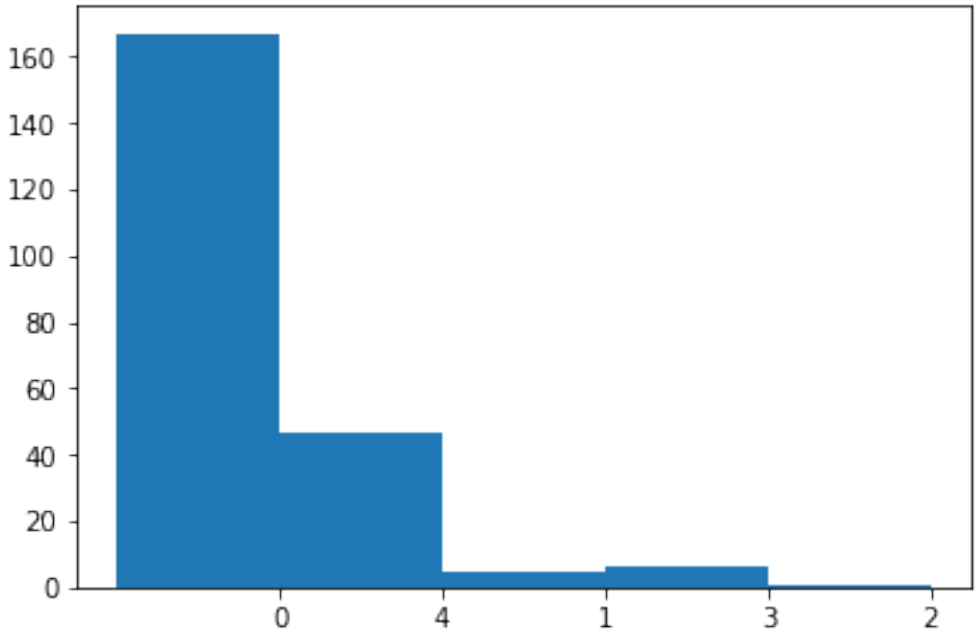
In [405]:
```
### experiment with different number of clusters
def experiemnt_with_different_cluster(k):
    from sklearn.cluster import KMeans
    model = KMeans(n_clusters=k)
    sou_clust = model.fit(we)
    y_means = sou_clust.predict(we)
    from collections import Counter

    labels, values = zip(*Counter(y_means).items())
    indexes = np.arange(len(labels))
    width = 1
    plt.bar(indexes, values, width)
    plt.xticks(indexes + width * 0.5, labels)
    plt.show()

    most_common = Counter(y_means).most_common(2)
    cluster1 = most_common[0][0]
    cluster2 = most_common[1][0]

    years = [int(x) for x in speeches['year']]
    cluster_year = [[int(x == y) for x in y_means] for y in range(10)]
    plt.scatter(years,cluster_year[cluster1],c = 'b')
    plt.scatter(years,cluster_year[cluster2],c = 'r')
```

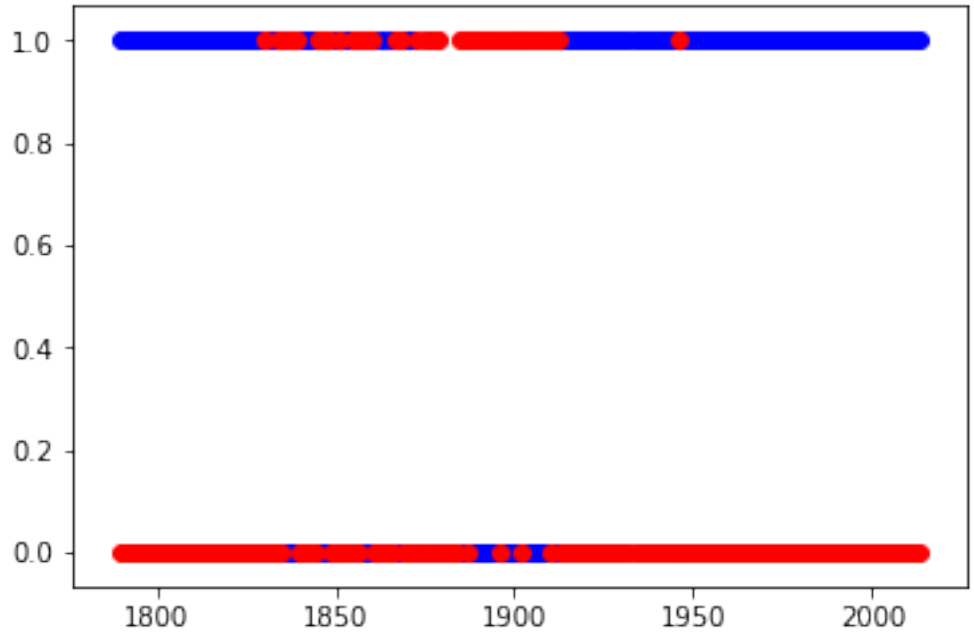
In [419]:
```
experiemnt_with_different_cluster(3)
```



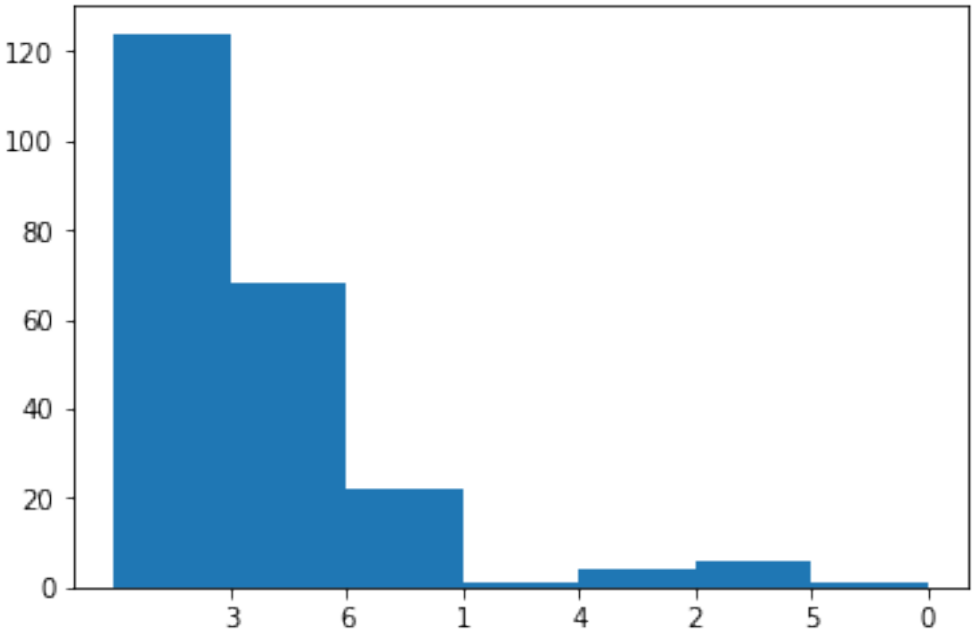


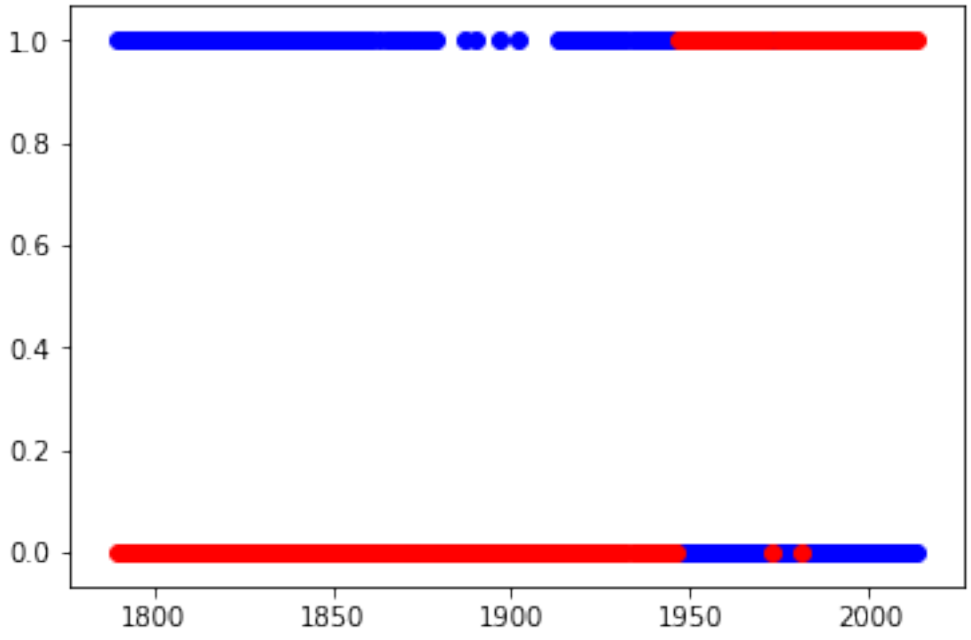

```
In [416]: experiemnt_with_different_cluster(5)
```



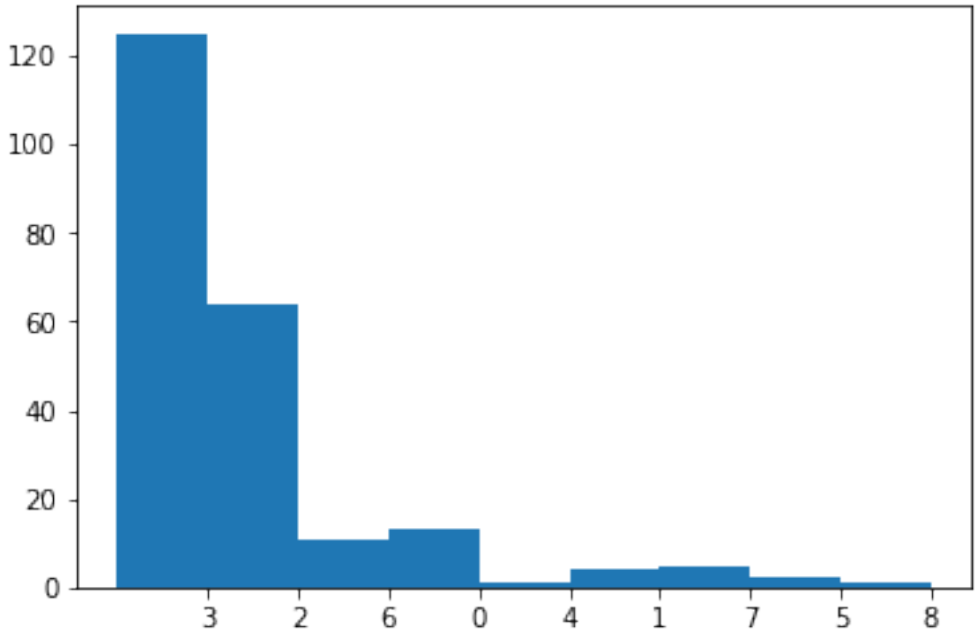


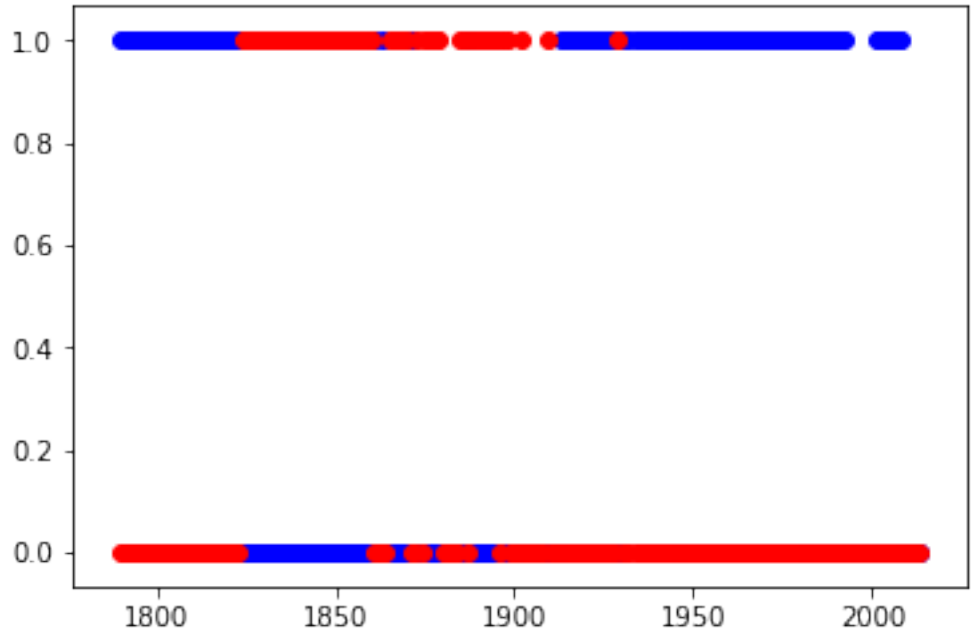

```
In [417]: experiemnt_with_different_cluster(7)
```






```
In [418]: experiemnt_with_different_cluster(9)
```





Analysis: From the various cluster numbers experimented, we can see there are mostly two main clusters, with very few left in other clusters; the two main clusters occur in alternatinbg fashion. It is also obvious that there is there seems to be dominating cluster related with time: for example, from the 9 cluster graph we can see clearly that after 1900, the second cluster almost

disappears with the first cluster. This can be interpreted that with the passing of time, wording tend to change in the political environment.

However, the KMeans clustering methods are not stable in this case. I get very different results every time I run it. Therefore, the trend and interpretation above might be dubious.