K Means

May 22, 2018

1 Homework 4 - K-Means Clustering

Here, we will explore the k-means clustering algorithm discussed in class, using the BBC dataset.

To help you remember and format your output, we've included some dummy data and cells. Feel free to remove those or add some more. Generally you'll want to keep the cells after "Step x.y Results".

1.1 Data Prep

Download the data set. There should be five files. Look at the readme to understand the use of each file. Load the data into a dataframe using any of the files you think are relevant. Columns should be terms and rows should be articles. The resulting dataset should have shape (2225, 9635). Create a pandas series for the class label as well. This will be used at the end of the assignment.

```
In [1]: import pandas as pd
        import sklearn
        import matplotlib.pyplot as plt
        import numpy as np
        import scipy.io
```

```
In [2]: # TODO: properly load the data instead of using the
        # dummy data
        matrix=scipy.io.mmread('bbc.mtx')
        matrix=matrix.todense(order=None, out=None)
        #matrix=matrix.transpose(matrix)
        bbc_df=pd.DataFrame(matrix)
        bbc_df=bbc_df.transpose()
        document_class=pd.read_csv('bbc.classes',delimiter=" ", skiprows=3)

        terms=pd.read_csv('bbc.terms',names=['col'])
        terms_list=terms['col'].values.tolist()
        bbc_df.columns=terms_list


        #document_class=document_class.as_matrix(columns='%Objects')
        document_class=document_class['%Objects'].values
        document_class=pd.Series(document_class)
```

```
#document_class=pd.series(document_class['col'])

#bbc_df = pd.DataFrame([{'ad': 1.0, 'sale': 5.0}])
#document_class = pd.Series([0.0])
#matrix
```

1.2 Step 1.1 Results

```
In [3]: bbc_df.head()
```

```
Out[3]:     ad  sale  boost  time  warner  profit  quarterli  media  giant  jump  \
        0  1.0   5.0    2.0   3.0     4.0    10.0        1.0    1.0    1.0   1.0
        1  0.0   0.0    1.0   2.0     0.0     0.0        0.0    0.0    0.0   0.0
        2  0.0   4.0    0.0   0.0     0.0     0.0        0.0    0.0    1.0   0.0
        3  0.0   1.0    0.0   0.0     0.0     4.0        1.0    0.0    0.0   0.0
        4  0.0   0.0    0.0   1.0     0.0     0.0        0.0    0.0    1.0   0.0

             ...     č339  denialofservic  ddo  seagrav  bot  wirelessli  streamcast  \
        0    ...      0.0             0.0  0.0      0.0  0.0         0.0         0.0
        1    ...      0.0             0.0  0.0      0.0  0.0         0.0         0.0
        2    ...      0.0             0.0  0.0      0.0  0.0         0.0         0.0
        3    ...      0.0             0.0  0.0      0.0  0.0         0.0         0.0
        4    ...      0.0             0.0  0.0      0.0  0.0         0.0         0.0

           peripher  headphon  flavour
        0       0.0       0.0      0.0
        1       0.0       0.0      0.0
        2       0.0       0.0      0.0
        3       0.0       0.0      0.0
        4       0.0       0.0      0.0

        [5 rows x 9635 columns]
```

```
In [4]: document_class
```

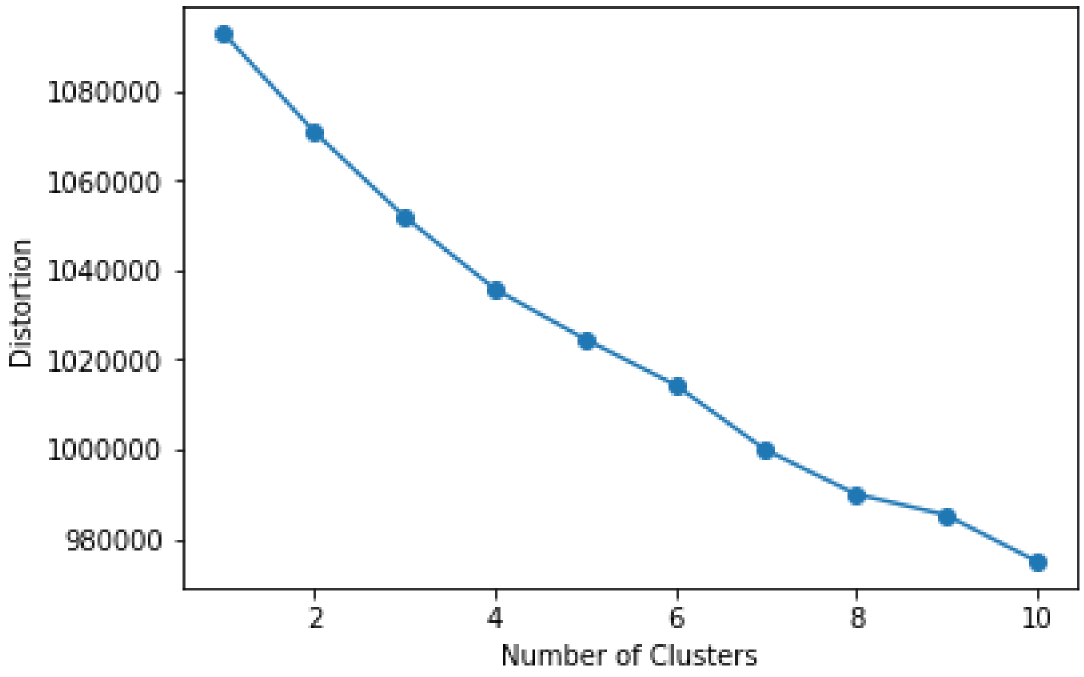
```
Out[4]: 0       0
        1       0
        2       0
        3       0
        4       0
        5       0
        6       0
        7       0
        8       0
        9       0
        10      0
        11      0
        12      0
        13      0
```

```
14      0
15      0
16      0
17      0
18      0
19      0
20      0
21      0
22      0
23      0
24      0
25      0
26      0
27      0
28      0
29      0
       ..
2195    4
2196    4
2197    4
2198    4
2199    4
2200    4
2201    4
2202    4
2203    4
2204    4
2205    4
2206    4
2207    4
2208    4
2209    4
2210    4
2211    4
2212    4
2213    4
2214    4
2215    4
2216    4
2217    4
2218    4
2219    4
2220    4
2221    4
2222    4
2223    4
2224    4
dtype: int64
```

1.3 Step 1.2: Computing K-Means Clusters

```
In [5]: from sklearn.cluster import KMeans
        bbc_df=bbc_df.as_matrix(columns=None)
        #km=KMeans(5,'random',n_init=300,max_iter=300,random_state=0)
        #km.fit(bbc_df)
        distortions =[]
        for i in range(1,11):
            km=KMeans(n_clusters=i,init='random',n_init=30,max_iter=300,random_state=0)
            km.fit(bbc_df)
            distortions.append(km.inertia_)
```

```
In [6]: # TODO: plot distortion, not these attributes!
        plt.plot(range(1,11),distortions,marker='o')
        plt.xlabel('Number of Clusters')
        plt.ylabel('Distortion')
        plt.show()

        D_distortions=distortions[1:10]
        #print(distortions)
        Distortions=['None','None']
        for i in range(9):
            Distortions.append(D_distortions[i])
        #plt.plot(bbc_df['ad'],bbc_df['sale'], c='blue', marker='o')
        #plt.show()
```



1.4 Step 1.2 Results

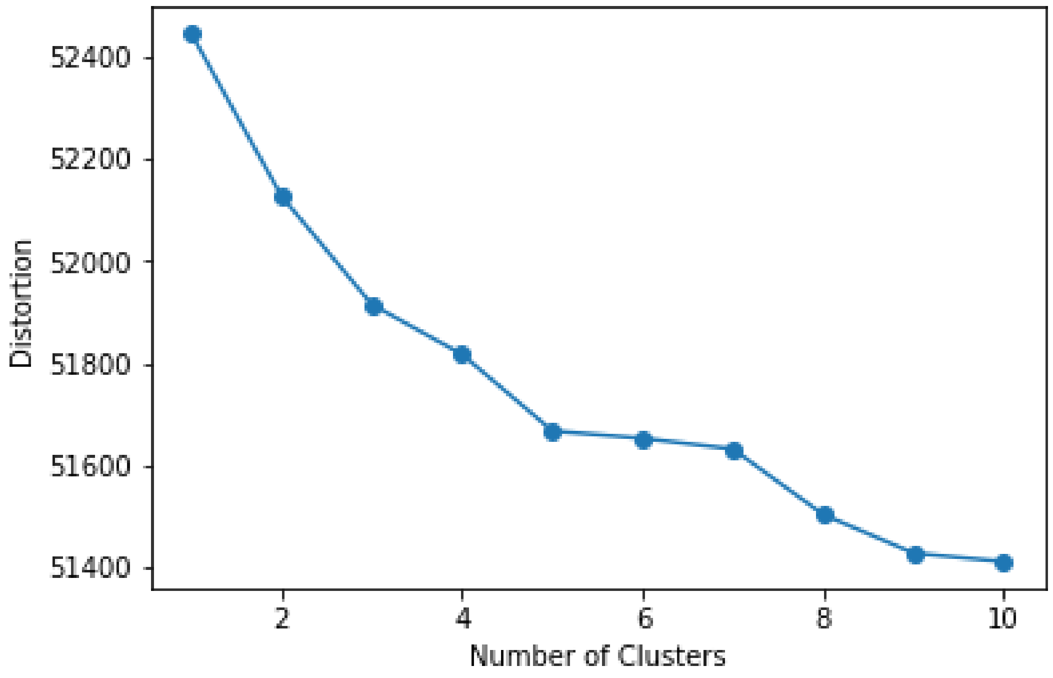
Update the first cell with just the numeric value of k and in the second cell output the distortions. The distortions should be a list where the index corresponds to the total distortion using that number of clusters (first two elements should be None since clustering using zero or one clusters doesn't make much sense). Example: [None, None, 100, 90, 80, 70, 60, 50, 40, 30, 20]. (Note that your list should have 11 elements.) 10

```
In [7]: # Output Distortions
        Distortions

Out[7]: ['None',
         'None',
         1070889.8681838671,
         1051938.5474288887,
         1035834.9386071024,
         1024659.7688110314,
         1014414.0501342831,
         999965.29646604729,
         990088.66911816795,
         985420.27479406528,
         975103.74528361089]
```

1.5 Step 1.3: Feature Scaling

```
In [8]: # TODO scale with min/max
        from sklearn import preprocessing
        min_max_scaler = preprocessing.MinMaxScaler()
        bbc_df_scaled = min_max_scaler.fit_transform(bbc_df)

In [9]: # TODO: update plot
        distortions =[]
        for i in range(1,11):
            km=KMeans(n_clusters=i,init='random',n_init=30,max_iter=300,random_state=0)
            km.fit(bbc_df_scaled)
            distortions.append(km.inertia_)



        plt.plot(range(1,11),distortions,marker='o')
        plt.xlabel('Number of Clusters')
        plt.ylabel('Distortion')
        plt.show()

        D_distortions=distortions[1:10]
        #print(distortions)
        Distortions=['None','None']
        for i in range(9):
            Distortions.append(D_distortions[i])
```

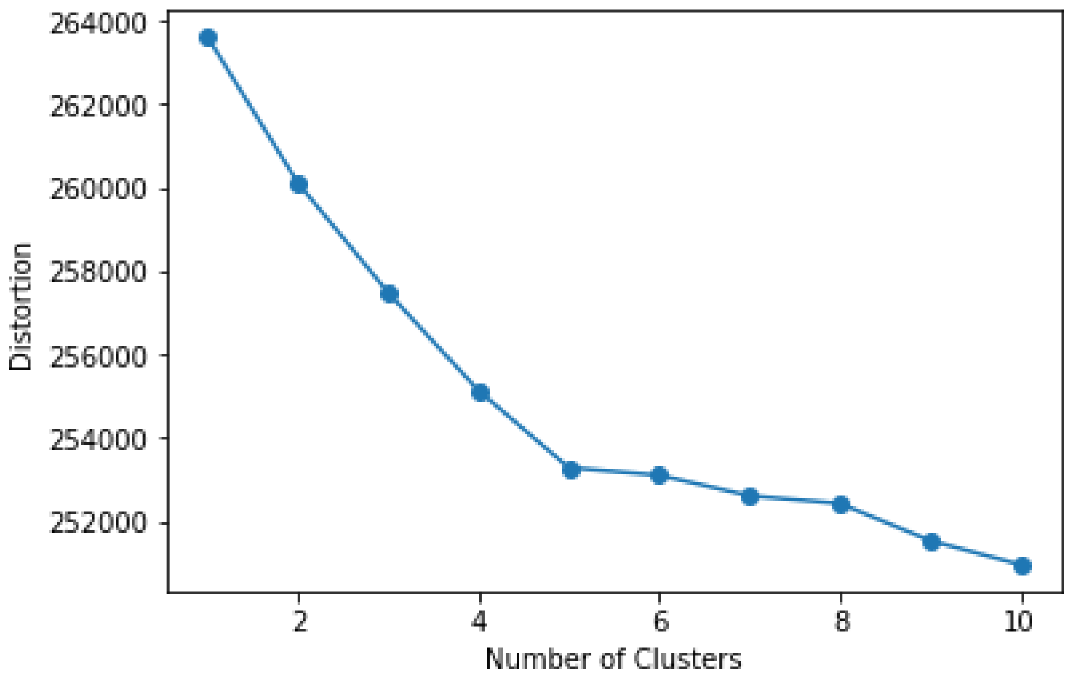
```
#plt.plot(bbc_df['ad'],bbc_df['sale'], c='blue', marker='o')
#plt.show()
```



```
In [10]: # TODO scale Boolean
         bbc_df[bbc_df!= 0] = 1
         bbc_boolean_maxtrix=bbc_df
```

```
In [11]: # TODO: update plot
         b_distortions =[]
         for i in range(1,11):
             km=KMeans(n_clusters=i,init='random',n_init=30,max_iter=300,random_state=0)
             km.fit(bbc_boolean_maxtrix)
             b_distortions.append(km.inertia_)


         plt.plot(range(1,11),b_distortions,marker='o')
         plt.xlabel('Number of Clusters')
         plt.ylabel('Distortion')
         plt.show()

         BD_distortions=b_distortions[1:10]
         #print(distortions)
         BDistortions=['None','None']
         for i in range(9):
             BDistortions.append(BD_distortions[i])
```

```
#plt.plot(bbc_df['ad'],bbc_df['sale'], c='blue', marker='o')
#plt.show()
```



1.6 Step 1.3 Results

Enter the value for k below. In the next two cells output the distortions of the scaled and then binary features follwoing the same instructions as above. 5

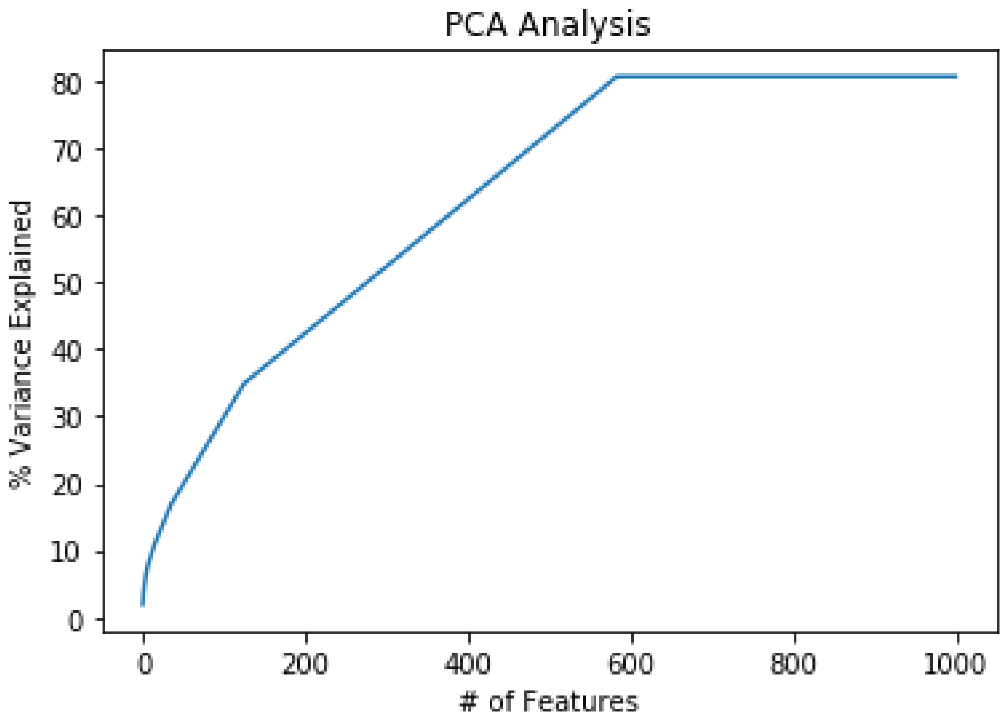
```
In [12]: # Output Scaled Distortions
         Distortions
```

```
Out[12]: ['None',
          'None',
          52126.321592112996,
          51914.913496705696,
          51816.863743916903,
          51667.194738693717,
          51652.812150477177,
          51631.921171240996,
          51503.498798472858,
          51427.900834439955,
          51411.311880990936]
```

```
In [13]: # Output Binary Distortions
         BDistortions

Out[13]: ['None',
          'None',
          260116.6405640911,
          257490.02839872747,
          255131.80428279497,
          253287.96490233613,
          253108.71214907069,
          252615.22843548973,
          252429.05559717535,
          251539.08186750309,
          250960.96576807476]
```

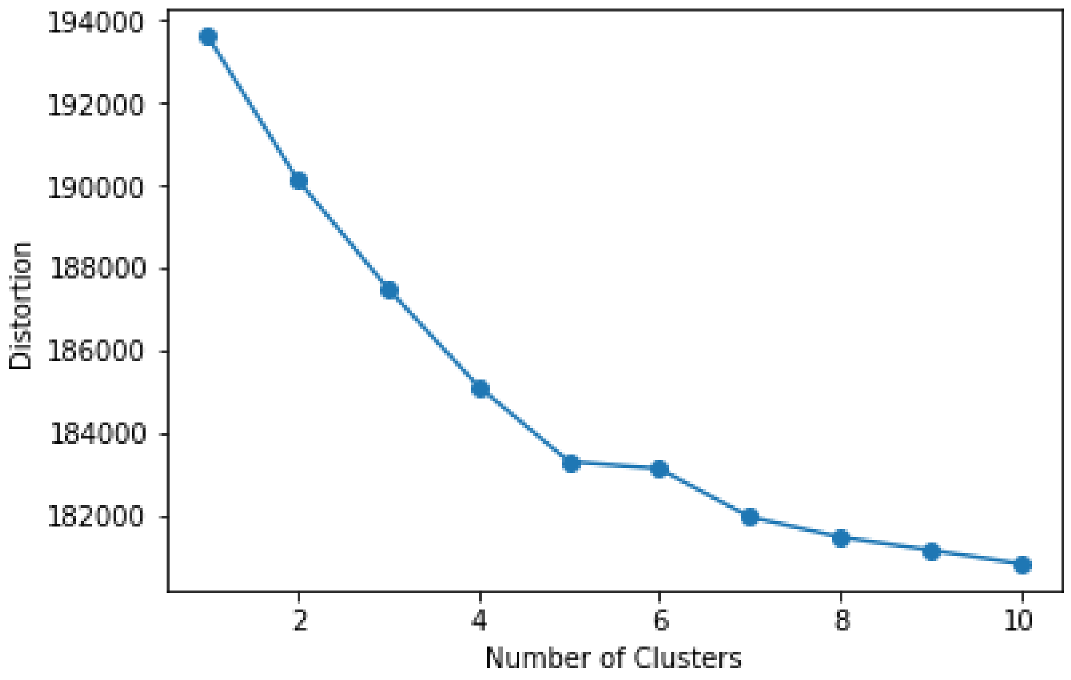
1.7 Step 1.4: Dimensionality Reduction via PCA

```
In [14]: from sklearn.decomposition import PCA
         pca_model = PCA(n_components = 1000)
         X_data=bbc_boolean_maxtrix
         pca_model.fit(X_data)
         variance = pca_model.explained_variance_ratio_
         cv=np.cumsum(np.round(pca_model.explained_variance_ratio_,decimals=3)*100)
         #X_PCA=pca_model.transform(X_data)
         # TODO: create cumulative sum of variance

In [15]: # TODO: Plot cv vs features
         #plt.plot(bbc_df['ad'],bbc_df['sale'], c='blue', marker='o')
         plt.ylabel('% Variance Explained')
         plt.xlabel('# of Features')
         plt.title('PCA Analysis')
         #plt.ylim(30,100.5)
         plt.style.context('seaborn-whitegrid')
         plt.plot(cv)
         plt.show()
```



```
In [16]: # TODO: Re-run PCA with the "correct" number of features and then run K-means in one th
         pca_model = PCA(n_components = 600)
         X_data=bbc_boolean_maxtrix
         pca_model.fit(X_data)
         variance = pca_model.explained_variance_ratio_
         cv=np.cumsum(np.round(pca_model.explained_variance_ratio_,decimals=3)*100)
         X_PCA=pca_model.transform(X_data)

         # K means
         pca_distortions =[]
         for i in range(1,11):
             km=KMeans(n_clusters=i,init='random',n_init=30,max_iter=300,random_state=0)
             km.fit(X_PCA)
             pca_distortions.append(km.inertia_)


         plt.plot(range(1,11),pca_distortions,marker='o')
         plt.xlabel('Number of Clusters')
         plt.ylabel('Distortion')
         plt.show()

         PCA_distortions=pca_distortions[1:10]
         #print(distortions)
         PCA_Distortions=['None','None']
```

```
for i in range(9):
    PCA_Distortions.append(PCA_distortions[i])
```



1.8 Step 1.4 Results

Enter the number of features in the next cell, then the value of k for the following cell. 6005

```
In [17]: # Output Distortions
         PCA_Distortions
```

```
Out[17]: ['None',
          'None',
          190128.9178736193,
          187500.29549195222,
          185121.64356752703,
          183313.29231894665,
          183136.46802803845,
          181971.15852348157,
          181478.70623677294,
          181161.85278419364,
          180833.60881083031]
```

1.9 Step 1.5: Evaluating Cluster Results

Rerun K-means with your best results and lets compare to the actual given class. One issue is that the cluster labels will not line up with your class names so we will need to map cluster labels to the class label

```
In [18]: from statistics import mode
         km=KMeans(5,init='random',n_init=30,max_iter=300,random_state=0)
         kmeans=km.fit(bbc_df_scaled)
         labels=km.labels_

In [19]: labels=km.labels_
         misclassi_count=0
         mis_index=[]
         for i in range(5):
             group_index=[]
             for index, values in enumerate(labels):
                 if values==i:
                     group_index.append(index)
             group_mode = mode(document_class[group_index])
             mis_classi=[]
             for index, value in enumerate(document_class[group_index]):
                 if value!=group_mode:
                     mis_classi.append(index)
                     mis_index.append(index)
             misclassi_count=misclassi_count+len(mis_classi)

         result=(misclassi_count)/np.size(bbc_df,0)
         print(result)
```

```
0.6530337078651686
```

Let's see how well the classifier works on the training data. Determine which data points are incorrectly labeled by the classifier.

```
In [20]: (document_class[mis_index])

Out[20]: 0       0
         1       0
         2       0
         3       0
         4       0
         5       0
         6       0
         7       0
         8       0
         9       0
         10      0
         11      0
         12      0
         13      0
```

```
14     0
15     0
16     0
17     0
18     0
19     0
20     0
21     0
22     0
23     0
24     0
25     0
26     0
27     0
28     0
29     0
      ..
583    1
584    1
585    1
586    1
587    1
588    1
589    1
590    1
591    1
592    1
593    1
594    1
595    1
596    1
597    1
598    1
599    1
600    1
601    1
602    1
603    1
604    1
605    1
606    1
607    1
608    1
609    1
610    1
611    1
612    1
dtype: int64
```

1.10 Step 1.5 Results

This time, enter the ratio between the number of misclassifications and the total number of rows.
0.6530337078651686