hw8_arkam2

March 28, 2021

1 STAT 542: Homework 8

1.1 NAME - ARKA MITRA (netid - arkam2)

1.1.1 Spring 2021, by Ruoqing Zhu (rqzhu)

1.1.2 Due: Tuesday, Mar 30, 11:59 PM CT

Question 1 [50 Points] Implement the K-Means Algorithm

Question 2 [50 Points] Clustering, Classification and Dimension Reduction

1.2 About HW8

We practice two main tools: PCA and k-means. For k-means, you need to code your own algorithm. For PCA, understand how to perform prediction using the rotation matrix and effectively reduce the dimension.

2 Question 1 [50 Points] Implement the K-Means Algorithm

Write your own code of k-means. The algorithm essentially performs the following:

Initialize cluster means or cluster membership
Iterate between the updates of cluster means and membership
Output the results when they do not change anymore

You should consider the following while writing the algorithm:

Avoiding over-use of for-loop such that your code can be applied to data with larger size (in question 2)
[10 points] You should further consider multiple starting values and compare the within-cluster distances to pick the best
[15 points] Compare your results with a built-in k means algorithm, e.g., `kmeans()`, on the `iris` data. Make sure that you understand the iris data. For both your code and the built-in function, use $k = 3$. Try 1 and 20 random starts and compare the results with different seeds. Do you observe any difference? What is the cause of the difference?

[21]:
```
import numpy as np
import pandas as pd
from sklearn import *
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist

def kmeans_cluster(X, k = 3, tol = 1e-08, n_starts = 10):
    '''
    K-means Clustering Algorithm

    Paramaters :

    - X (Input)                : features to be clustered (n x p)
    - k (Input)                : no. of clusters
    - tol (Input)              : tolerance to declare convergence
    - n_starts (Input)         : number of random starts for cluster centers
                                 iteration with minimum within-cluster distance␣
↪is output
    - clusters_ (Output)       : array of clusters (n x 1)
    - centers_  (Output)       : cluster centers (k x p)
    - clust_dist (Output)      : sum of squared distances to closest centroid

    '''
    clust_dist = 1e08

    for run in range(n_starts):

        center_ind = np.random.randint(0, len(X), size = k)
        centers = X[center_ind]

        tmp_centers = np.zeros([centers.shape[0], centers.shape[1]])

        while np.linalg.norm(tmp_centers - centers) > tol:
            distmat = cdist(X, centers)
            clusters = np.argmin(distmat, axis = 1)
            tmp_centers = np.copy(centers)
            for i in range(k):
                centers[i] = np.mean(X[clusters == i] , axis = 0 )

        within_clust_dist = np.sum((np.min(distmat, axis = 1))**2 )

        if within_clust_dist < clust_dist:
            clusters_ = np.copy(clusters)
            centers_ = np.copy(centers)
            clust_dist = within_clust_dist

    return clusters_, centers_, clust_dist

iris = datasets.load_iris()
X, y = iris.data , iris.target
```

```
print('K-Means Cluster from our algorithm :')
print(kmeans_cluster(X, k = 3, tol = 1e-08, n_starts = 10))

km = KMeans(n_clusters = 3, n_init=10).fit(X)
print('K-Means Cluster from Python sci-kit K-Means algorithm :')
print([km.labels_ , km.cluster_centers_ , km.inertia_])
```

```
K-Means Cluster from our algorithm :
(array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 2, 2, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0,
       0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2], dtype=int64),
array([[6.85      , 3.07368421, 5.74210526, 2.07105263],
       [5.006     , 3.428     , 1.462     , 0.246     ],
       [5.9016129 , 2.7483871 , 4.39354839, 1.43387097]]), 78.85144142614601)
K-Means Cluster from Python sci-kit K-Means algorithm :
[array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2,
       2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1]), array([[5.006
, 3.428     , 1.462     , 0.246     ],
       [5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]]), 78.851441426146]
```

The cluster centers and sum of squared distances of samples to their closest cluster center is exactly equal in both our code and Python sklearn k-means clustering.

[23]:
```
print('K-Means Cluster from our algorithm :')
print(kmeans_cluster(X, k = 3, tol = 1e-08, n_starts = 1))

print('K-Means Cluster from our algorithm :')
print(kmeans_cluster(X, k = 3, tol = 1e-08, n_starts = 20))
```

```
K-Means Cluster from our algorithm :
(array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1], dtype=int64),
array([[6.85384615, 3.07692308, 5.71538462, 2.05384615],
       [5.88360656, 2.74098361, 4.38852459, 1.43442623],
       [5.006     , 3.428     , 1.462     , 0.246     ]]), 78.8556658259773)
K-Means Cluster from our algorithm :
(array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1], dtype=int64),
array([[6.85      , 3.07368421, 5.74210526, 2.07105263],
       [5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [5.006     , 3.428     , 1.462     , 0.246     ]]), 78.85144142614601)
```

K-Means algorithm is sensitive to the choice of initial clusters.

3 Question 2 [50 Points] Clustering, Classification and Dimension Reduction

Although clustering is an unsupervised algorithm, it may sometimes reveal the underlying true (unobserved) label associated with each observation. For this question, apply your K means clustering algorithm on the handwritten digit data (`zip.train` from the `ElemStatLearn` package). We only use digits 1, 4 and 8.

Given your clustering results, can you propose a method to assign class labels to each of your clusters?
With your assigned cluster labels, how to predict labels on the `zip.test` data?
What is the classification error based on your model? A classification error is defined as $\frac{1}{n}\sum_i 1(y_i \neq \hat{y}_i)$.

Dimensionality often causes problems in a lot of machine learning algorithms. PCA is an effective way to reduce the dimension without sacrificing the clustering/classification performances. Repeat your analysis by

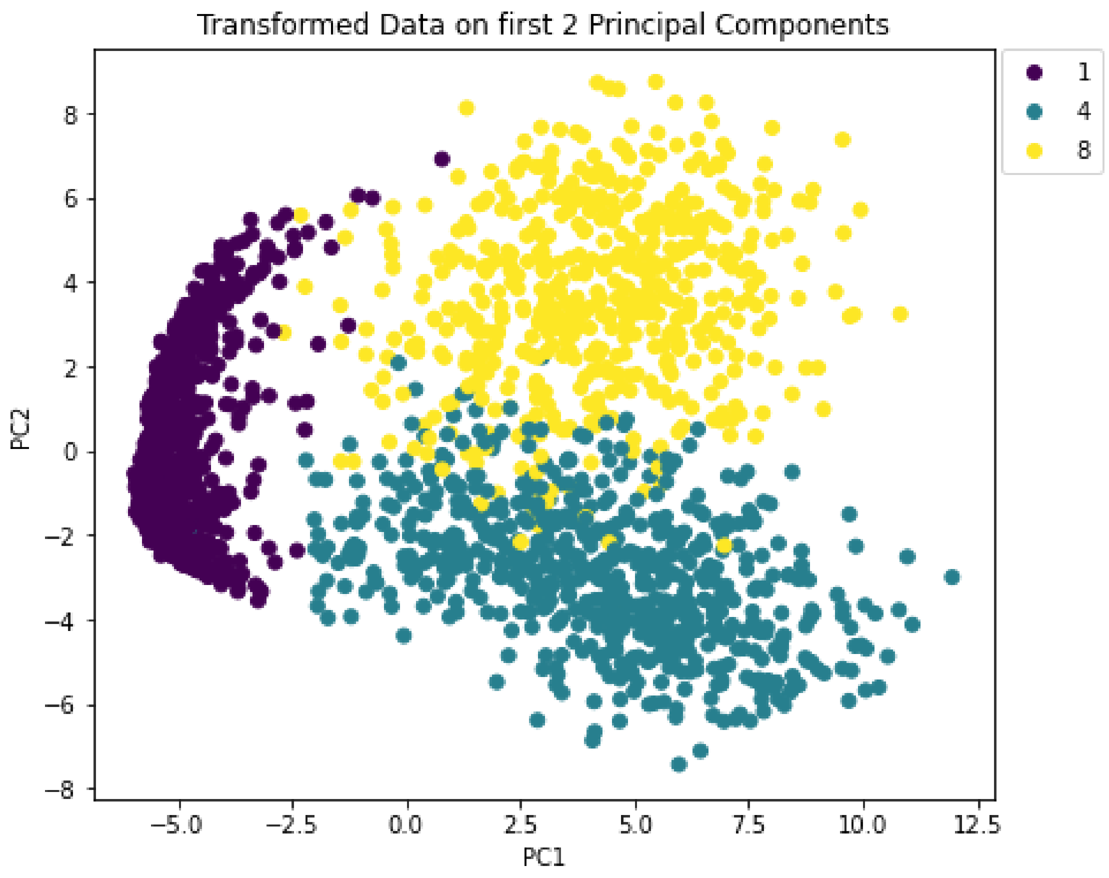
Process your data using PCA. Plot the data on a two-dimensional plot using the first two PC's. Mark the data points with different colors to represent their digits.
Based on the first two PCs, redo the k-means algorithm. Again, assign each cluster a label of the digit, and predict the label on the testing data. You need to do this prediction properly, meaning that your observed testing data is still the full data matrix, and you should utilize the information of PCA from the training data to construct new features of the testing data.
Compare your PCA results with the results obtained by using the entire dataset. How much classification accuracy is sacrificed by using just two dimensions?
Comment on the potential strength and drawback of this dimension reduction approach compared with the raw data approach.

3.1 Solution

`zip.train` and `zip.test` has been saved as txt files from ElemStatLearn, and used here.

```
[26]: import matplotlib.pyplot as plt
      from sklearn.decomposition import PCA

      digits_train , digits_test = np.loadtxt('zip.train.txt') , np.loadtxt('zip.test.
       ↪txt')
      train_df , test_df = pd.DataFrame(digits_train) , pd.DataFrame(digits_test)
      train_df = train_df[(train_df[0] == 1.0) | (train_df[0] == 4.0) | (train_df[0]␣
       ↪== 8.0)]
      test_df = test_df[(test_df[0] == 1.0) | (test_df[0] == 4.0) | (test_df[0] == 8.
       ↪0)]

      #Split the data into training and test
      X_train, y_train = train_df.values[:,1:], train_df.values[:,0]
      X_test, y_test  = test_df.values[:,1:], test_df.values[:,0]

      kmeans = kmeans_cluster(X_train, k = 3)
      clusters , cluster_centers = list(kmeans[0]) , kmeans[1]

      labels = [1.0, 4.0, 8.0]
      mapmat = np.zeros([len(labels),k])
      for i in range(len(labels)):
          for j in range(k):
              mapmat[i,j] = np.linalg.norm(np.mean(X_train[train_df[0] == labels[i]],␣
       ↪axis = 0) - cluster_centers[j])
      print(np.round(mapmat , 2))

      mapdict = {}
      for i in range(k):
          mapdict[i] = labels[ np.argmin(mapmat , axis = 0)[i] ]
      print("Labels:" , mapdict)


      distmat = cdist(X_test, cluster_centers)
      predicted_clusters = np.argmin(distmat, axis = 1)
      predicted_labels = [mapdict[k] for k in predicted_clusters]

      err = sum(predicted_labels != y_test)/len(y_test)
      print("Classification Error = ", np.round(err, 3))

      pca = PCA(n_components = 2).fit(X_train)
      X_train_pca = pca.transform(X_train)

      plt.figure(figsize = (7,6))
      scatter = plt.scatter(X_train_pca[:,0], X_train_pca[:,1] , c = y_train)
      plt.legend(*scatter.legend_elements(), bbox_to_anchor=(1.01, 1), loc='upper␣
       ↪left', borderaxespad=0)
      plt.title('Transformed Data on first 2 Principal Components')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

```
[[ 0.25 10.15 10.13]
 [ 9.54  7.23  0.45]
 [ 9.75  0.35  7.1 ]]
Labels: {0: 1.0, 1: 8.0, 2: 4.0}
Classification Error =  0.056
```



Each cluster is labelled using the cluster centers and training labels.

To predict a label on the test data, we predict the cluster using obtained cluster centers and see how much they differ from the obtained cluster centers.

[27]:
```
km_pca = kmeans_cluster(X_train_pca, k = 3)
clusters_pca , cluster_centers_pca = list(km_pca[0]) , km_pca[1]

#Mapping
labels = [1.0, 4.0, 8.0]
```

```
mapmat_pca = np.zeros([len(labels),k])
for i in range(len(labels)):
    for j in range(k):
        mapmat_pca[i,j] = np.linalg.norm(np.mean(X_train_pca[train_df[0] ==␣
 ↪labels[i]], axis = 0)
                                          - cluster_centers_pca[j])
print(mapmat_pca)
print('\n')

mapdict_pca = {}
for i in range(k):
    mapdict_pca[i] = labels[ np.argmin(mapmat_pca , axis = 0)[i] ]
print("Labels:" , mapdict_pca)

#Transform Xtest
X_test_pca = pca.transform(X_test)

#Prediction
distmat_pca = cdist(X_test_pca, cluster_centers_pca)
predicted_clusters_pca = np.argmin(distmat_pca, axis = 1)
predicted_labels_pca = [mapdict_pca[k] for k in predicted_clusters_pca]


pca_error = sum(predicted_labels_pca != y_test)/len(y_test)
print("The Classification Error = ", np.round(pca_error, 3))
```

```
[[10.13345291  9.91570388  0.21954723]
 [ 0.49269132  6.77675472  9.42341232]
 [ 6.69603331  0.22225714  9.63860931]]


Labels: {0: 4.0, 1: 8.0, 2: 1.0}
The Classification Error =  0.083
```

Classification Error using all features is 0.056, but with only first 2 PCs is 0.083. Only ~0.03 accuracy is sacrificed by using just 2 PCs.

The strengths and weaknesses of PCA ties to its non-parametric nature. Given, there are no parameters to estimate or adjust and hence, results are data-driven, not dependent on user choices. However, it does not account for any a-priori knowledge, as the parametric algorithms do. This might be unacceptable for many applications.

```
[ ]:
```