AU311, Pattern Recognition Tutorial (Fall 2017) Homework: 3. Unsupervised Learning

3. Unsupervised Learning

Lecturer: Xiaolin Huang xiaolinhuang@sjtu.edu.cn

Student: Houlin Li lihoulin@sjtu.edu.cn

Problem 1

For data set "Boston Housing" (https://blog.csdn.net/qq_41185868/article/details/87691801), which contains 14 features. Please

i) calculate the Pearson linear correlation between the housing price and each of the features;

ii) use PCA to find the principal components;

iii) calculate the Pearson linear correlation between the housing price and the first three principle components.

Ans:

i)According to the definition of Pearson linear correlation. I have the codes below:

```
data = pd.read_table('housing.data',header=None,delim_whitespace=True)
data = np.array(data)
label = data[:, 13:14]
feature = data[:, 0:13]

plcs = []
y = data[:,-1]
for i in range((data.shape[1]-1)):
    x = data[:,i]
    #print(x)
    plc = np.corrcoef(x, y)
    #print(plc)
    plcs.append(plc[0,1])

print(plcs)
```

So, I get the plc between each of the features and price is:

```
[-0.3883046085868113, 0.36044534245054277, -0.4837251600283727,
0.17526017719029854, -0.42732077237328264, 0.6953599470715395,
-0.3769545650045963, 0.24992873408590388, -0.38162623063977763,
-0.468535933567767, -0.5077866855375617, 0.33346081965706653,
-0.7376627261740147]
```

Figure 1: PLC

ii)

PCA algorithm mainly includes the following steps:

- Centralize all samples.
- Calculate covariance matrix.
- Eigenvalue decomposition of covariance matrix.
- Select the largest n eigenvalues and their corresponding eigenvectors.

the codes are below:

```
#Centralize all samples.
mean = feature.mean(axis=0)
newdata_x = (feature - mean) / feature.std()
print(newdata_x.shape)
#print(feature.mean(axis=0))
#print(feature - feature.mean(axis=0))
#Calculate covariance matrix.
covMat=np.cov(newdata_x,rowvar=0)
#print(covMat)
#Eigenvalue decomposition of covariance matrix.
eigVals,eigVects=np.linalg.eig(np.mat(covMat))
#print(eigVals)

#Select the largest n eigenvalues and their corresponding eigenvectors.
index = np.argsort(-eigVals)
n_eigVect=eigVects[:,0:3]
print(n_eigVect)
```

As we know, the principal components is the eigenvector corresponding to the maximum eigenvalue, so it is below:

```
[ 2.92973218e-02
[-4.35898000e-02
[ 2.83309382e-02
[-5.55846350e-05
[ 4.49721818e-04
[-1.16815860e-03
[ 8.36335746e-02
[-6.56163360e-03
[ 4.50053753e-02
[ 9.49741169e-01
[ 5.60011721e-03
[-2.91218514e-01
[ 2.29433756e-02
```

iii)

After selecting the feature vectors corresponding to the three largest eigenvalues, I project the original data onto these three feature vectors. Then calculate the new data and price to get the Pearson linear correlation. The codes are below:

```
recentdata = np.matmul(feature, n_eigVect)
plc2 = []
for i in range((recentdata.shape[1])):
    x = recentdata[:,i]
    x = np.squeeze(x)
    #print(x)
    plc = np.corrcoef(x, y)
    #print(plc)
    plc2.append(plc[0,1])
print(plc2)
```

So the Pearson linear correlation between the housing price and the first three principle components is :

```
[-0.4869780582650924, -0.07529854896897437, -0.21186333427007786]
```

Problem 2

Implement K-means clustering method on the data set “fisheriris” (matlab build-in dataset). Try different strategies and then use the label to measure the clustering accuracy.

Ans:

The code of K-means algorithm is as follows:

```
import numpy as np
import pandas as pd
import math
import random

columns = ['1','2','3','4','Category']
data = pd.read_table('iris.data',header=None,sep=',',names=columns)

#print(data)
data['Category'] = data['Category'].map({'Iris-setosa':0, 'Iris-versicolor':1, 'Iris-virginica' :2}
                                                                                )
data = np.array(data)
#print(data)
data_x = data[:,0:4]
data_y = data[:,-1]
print(data_y)

#distance
def distance(vec1,vec2):
    return np.sqrt(np.sum((vec1-vec2)**2))

#randly chose the centers
def chosecent(data,k):
    m,n = data.shape
    center = np.zeros((3,n))
    index1 = int(np.random.uniform(0,50))
    index2 = int(np.random.uniform(50,100))
    index3 = int(np.random.uniform(100,150))

    center[0,:] = data[index1,:]
```

```
    center[1,:] = data[index2,:]
    center[2,:] = data[index3,:]
    return center

##Kmeans
def Kmeans(data,k):
    m,n = data.shape
    #Record the category after clustering
    clustertabel = np.zeros((m,2))
    #Set whether the cluster category is changed as a sign of cycle termination
    clusterchange = True
    #Acquisition Center
    center = chosecent(data,k)
    while clusterchange:
        clusterchange = False
        #Traverse each sample to get its category
        for i in range(m):
            mindistance = 100000
            mincluster = -1
            for j in range(k):
                newdistance = distance(data[i,:], center[j, :])
                newcluster = j
                if newdistance < mindistance:
                    mindistance = newdistance
                    mincluster = newcluster
            #If the category changes, continue to enter the cycle and recalculate the cluster
                                                        center
            if mincluster != clustertabel[i,0]:
                clusterchange = True
                clustertabel[i,:] = mincluster,mindistance**2
        for j in range(k):
            pointsInCluster = data[np.nonzero(clustertabel[:,0] == j)[0]]  # Get all points of the
                                                        cluster class
            center[j,:] = np.mean(pointsInCluster,axis=0)   # Average the rows of a matrix
    return center, clustertabel

m = data.shape[0]
#print(m)
clustertabel = np.zeros((m,2))
center, clustertabel = Kmeans(data, 3)
print(clustertabel[:,0])
yes = np.count_nonzero(clustertabel[:,0]==data_y)
#print(yes)
accuracy = yes/m
print(accuracy)
```

The accuracy rate obtained through multiple random selection of the center value is 0.99333.