Spring 2019 - Intro to Machine Learning (CS-580L-01)
HOMEWORK-04 SOLUTION

Antra Aruj | aaruj1@binghamton.edu

Problem Statement:

This homework provides an exercise on K-mean algorithm. Use the K-means algorithm to cluster the following eight 2-dimensional data points into three clusters. Use Euclidean distance as the distance function, and (2,3), (3,3), and (5,4) as the initial centroids (means). Show detailed steps. For each iteration, show the clusters formed, the new means, and also the Sum of Squared Error as defined in the lecture slides. (You are highly recommended to write a program to do the computation and submit well-commented source code and program output of each iteration as the required detailed steps.)

X	Y
1	1
1	2
2	1
2	3
3	3
4	5
5	4
6	5

Implementation:

I have implemented the k-mean algorithm is python using pandas, NumPy, and matlibplot libraries. The code is segmented to follow the natural order of the k-mean algorithm. Firstly, I load the co-ordinates and initial guesses for centroids. Then, I calculate the Euclidean distance of each point from these three centroids and cluster them around respective centroids. Once the clusters are formed I calculate the sum squared error (SSE) to assess the accuracy of clustering. In the next step, I recalculate the centroid of each cluster. The above four steps are repeated until the centroid stop changing.

NOTE:

I have developed the homework in both python 2 and 3 and included them in the submission with file name "k-means_python_2.7.py" & "k-means_python_3.6.py". Python 3 version code doesn't work properly in remote machine therefore please use python 2 in remote machine.

RESULTS:

Cluster and Points in 1st Iteration	Cluster and Points in 2nd Iteration	Cluster and Points in 3rd Iteration
x y cluster 0 1 1 c1 1 1 2 c1 2 2 1 c1 3 2 3 c1 4 3 3 c2 5 4 5 c3 6 5 4 c3 7 6 5 c3	x y cluster 0 1 1 c1 1 1 2 c1 2 2 1 c1 3 2 3 c2 4 3 3 c2 5 4 5 c3 6 5 4 c3 7 6 5 c3	x y cluster 0 1 1 c1 1 1 2 c1 2 2 1 c1 3 2 3 c2 4 3 3 c2 5 4 5 c3 6 5 4 c3 7 6 5 c3

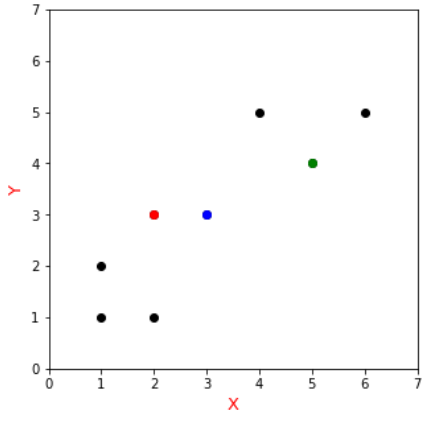
a. **Sum of Squared Error (in iteration 0): 15.00**
b. **Sum of Squared Error (in iteration 1): 5.60**
c. **Sum of Squared Error (in iteration 2): 4.50**

1st Iteration	2nd Iteration	3rd Iteration
Modified Centroid : index x_c y_c 0 1.50 1.75 1 3.00 3.00 2 5.00 4.67	Modified Centroid : index x_c y_c 0 1.33 1.33 1 2.50 3.00 2 5.00 4.67	Modified Centroid : index x_c y_c 0 1.33 1.33 1 2.50 3.00 2 5.00 4.67

The details of implementation are provided below:

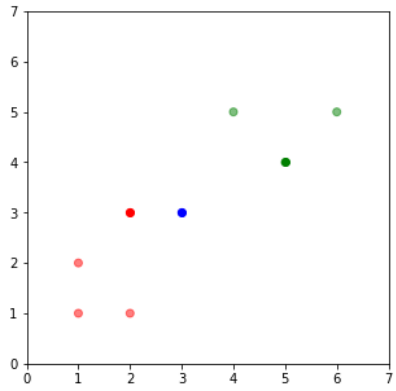
1) Plotting the given coordinates and centroid

```
x_coord = np.array([1,1,2,2,3,4,5,6])
y_coord = np.array([1,2,1,3,3,5,4,5])

x_centroid = np.array([2.0,3.0,5.0])
y_centroid = np.array([3.0,3.0,4.0])

plt.figure(figsize=(5, 5))
plt.scatter(x_coord, y_coord, color='k')

color_array = np.array(['r','b','g'])
for i in range(len(x_centroid)):
    plt.scatter(x_centroid[i], y_centroid[i] ,color = color_array[i])

plt.xlim(0, 7)
plt.ylim(0, 7)
plt.xlabel('X', fontsize='13', color='r')
plt.ylabel('Y', fontsize='13', color='r')
plt.show()
```



2) Calculating the Euclidean distance from centroids

```
def distance(x_coord, y_coord, x_centroid, y_centroid):
#     print('EUCLIDEAN DISTANCE FROM CENTROID :\n')
    dist_mat = np.zeros(shape=(len(x_coord),len(x_centroid)))
    for i in range(len(x_centroid)):
        for j in range(len(x_coord)):
            dist_mat[j,i] = np.sqrt((x_centroid[i] - x_coord[j])**2 + (y_centroid[i] - y_coord[j])**2)

    dataframe = pd.DataFrame(dist_mat)

    distance_columns = dataframe.columns =  ['c1','c2', 'c3']
    dataframe['cluster'] = dataframe.loc[:, distance_columns].idxmin(axis=1)

    for i in range(len(color_array)):
        dataframe.loc[dataframe["cluster"] == 'c{}'.format(i+1), "color"] = color_array[i]

    dataframe['x'] = x_coord
    dataframe['y'] = y_coord

    print(dataframe[['x','y', 'cluster']])

    plt.figure(figsize=(5, 5))
    plt.scatter(x_coord, y_coord, color=dataframe['color'], alpha=0.5)
    for i in range(len(x_centroid)):
        plt.scatter(x_centroid[i], y_centroid[i] ,color = color_array[i])

    plt.xlim(0, 7)
    plt.ylim(0, 7)
    plt.show()
    return dataframe
```



Initial Euclidean distance table from given centroids

Index	c1	c2	c3	cluster	color	x	y
0	2.23607	2.82843	5	c1	r	1	1
1	1.41421	2.23607	4.47214	c1	r	1	2
2	2	2.23607	4.24264	c1	r	2	1
3	0	1	3.16228	c1	r	2	3
4	1	0	2.23607	c2	b	3	3
5	2.82843	2.23607	1.41421	c3	g	4	5
6	3.16228	2.23607	0	c3	g	5	4
7	4.47214	3.60555	1.41421	c3	g	6	5

Cluster and Points in 1st Iteration	Cluster and Points in 2nd Iteration	Cluster and Points in 3rd Iteration
x y cluster 0 1 1 c1 1 1 2 c1 2 2 1 c1 3 2 3 c1 4 3 3 c2 5 4 5 c3 6 5 4 c3 7 6 5 c3	x y cluster 0 1 1 c1 1 1 2 c1 2 2 1 c1 3 2 3 c2 4 3 3 c2 5 4 5 c3 6 5 4 c3 7 6 5 c3	x y cluster 0 1 1 c1 1 1 2 c1 2 2 1 c1 3 2 3 c2 4 3 3 c2 5 4 5 c3 6 5 4 c3 7 6 5 c3

3) Calculating the Sum of Squared Error from centroids

```
def ss_error():
    sum_error = 0.00
    for i in range(len(x_centroid)):
        cluster_size = len(dataframe[dataframe["cluster"] == 'c{}'.format(i+1)])
        x_temp = dataframe.loc[dataframe["cluster"] == 'c{}'.format(i+1)]['x'].values
        y_temp = dataframe.loc[dataframe["cluster"] == 'c{}'.format(i+1)]['y'].values
        for j in range(cluster_size):
            sum_error = sum_error + ((x_temp[j] - x_centroid[i]) **2
                                     + (y_temp[j] - y_centroid[i])**2)
    print('Sum of Squared Error : ', "%0.2f" %sum_error)
    print('\n\n')
```

a. **Sum of Squared Error (in iteration 0): 15.00**
b. **Sum of Squared Error (in iteration 1): 5.60**
c. **Sum of Squared Error (in iteration 2): 4.50**

4) Updating the centroid by calculating the mean of clusters

```
def updated_centroid(x_centroid,y_centroid):
    print('Modified Centroid :','\n')
    print('index', '\t ', 'x_c', '\t ', 'y_c', '\n')

    for i in range(len(x_centroid)):
        x_centroid[i] = np.mean(dataframe.loc[dataframe["cluster"] == 'c{}'.format(i+1)]['x'])
        y_centroid[i] = np.mean(dataframe.loc[dataframe["cluster"] == 'c{}'.format(i+1)]['y'])

        print(i, '\t ', "%0.2f" %x_centroid[i], '\t ', "%0.2f" % y_centroid[i], '\n')
```

1st Iteration	2nd Iteration	3rd Iteration
Modified Centroid :	Modified Centroid :	Modified Centroid :
index x_c y_c	index x_c y_c	index x_c y_c
0 1.50 1.75	0 1.33 1.33	0 1.33 1.33
1 3.00 3.00	1 2.50 3.00	1 2.50 3.00
2 5.00 4.67	2 5.00 4.67	2 5.00 4.67

5) Iterating the K-Means until the old centroid and current centroid are equal

```
while True:

    dataframe = distance(x_coord, y_coord, x_centroid, y_centroid)
    ss_error()
    old_x_centroids = copy.deepcopy(x_centroid)
    old_y_centroids = copy.deepcopy(y_centroid)
    updated_centroid(x_centroid, y_centroid)
    if (np.array_equal(old_x_centroids, x_centroid) and np.array_equal(old_y_centroids, y_centroid)):
        break

plt.figure(figsize=(5, 5))
plt.scatter(x_coord, y_coord, color=dataframe['color'], alpha=0.5)
plt.xlim(0, 7)
plt.ylim(0, 7)
plt.xlabel('X', fontsize='13', color='r')
plt.ylabel('Y', fontsize='13', color='r')
plt.show()
```

6) Plotting K-Means final result