

BOĞAZIÇI UNIVERSITY

NONLINEAR MODELS IN OPERATIONS RESEARCH
IE 440

Final Question 1 - Final Project

Author:
Sefa Kayraklık

4 January 2020



Department of Industrial Engineering
Boğaziçi University

INTRODUCTION

The project is implemented using Python as the programming language. In the first part of the project, the given data is clustered by using K-mean clustering method with batch mode and on-line mode and self organizing map (SOM) method. In its second part the traveling salesman problem (TSP) is solved by using SOM method.

The source code used to import the data:

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5
6 data = pd.read_csv('IE440Final19ClusteringData.
7   txt', sep='\t', header=None, names=['x', 'y',
8   'class']);
9 data = data.drop(data.index[0])
10 data_tsp = pd.read_csv('IE440Final19ETSPData.txt',
11   sep=',', header=None, names=['City', 'x', 'y']);
12 data_tsp = data_tsp.drop(data_tsp.index[0])
```

Some useful functions for plot construction:

```
1 def plotClusteringGraph(data, w_star, b_star,
2   title, name="graph"):
3   if isinstance(data, np.ndarray):
4     patterns = data
5   else:
6     patterns = np.array(data[['x', 'y']], dtype=
7     np.float)
8   N = np.size(b_star, 1)
9   colors = ['tab:blue', 'tab:orange', 'tab:green',
10   'tab:red', 'tab:purple', 'tab:brown', 'tab:
11   pink', 'tab:gray', 'tab:olive', 'tab:
12   cyan', 'tab:brown', 'tab:orange', 'tab:
13   green', 'tab:red', 'tab:cyan']
14   plt.figure()
15   for n in range(N):
16     plt.scatter(w_star[n,0], w_star[n,1], marker
17     = "*", s=250, label=n+1, color=colors[n])
18     plt.scatter(patterns[b_star[:,n]==1,0],
19     patterns[b_star[:,n]==1,1], color=colors
20     [n], alpha=0.3)
21   plt.title(title)
22   plt.savefig("{0}.png".format(name))
23
24 def cal_dist(data):
25   data = data[:, [0,1]]
26   N = np.size(data, 0)
27   total=0
28   for n in range(N-1):
29     total += np.linalg.norm(data[n+1,:]-data[n
30     ,:])
31   total += np.linalg.norm(data[N-1,:]-data[0,:])
32   return total
33
34 def plotSOM_TSPGraph(data, w_star, b_star, title,
35   name="graph"):
36   patterns = data[:, [0,1]]
37   N = np.size(b_star, 1)
```

```
27 plt.figure()
28 for n in range(N):
29   plt.scatter(w_star[n,0], w_star[n,1], s=20,
30   label=n+1, color="red")
31 plt.plot(patterns[:,0], patterns[:,1], 'bo-')
32 plt.title(title)
33 plt.savefig("{0}.png".format(name))
```

I. CLUSTERING

The following three methods, K-mean clustering with batch mode and on-line mode, and (SOM), are considered to cluster the given data that are shown in the Figure 1.

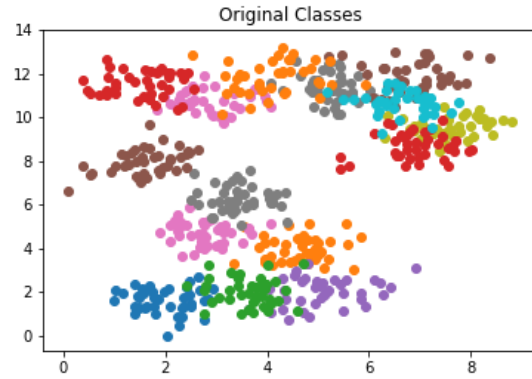


Fig. 1. The original data and its clusters

A. K-mean clustering with batch mode:

In this method, the initial centers are selected randomly and the members of the clusters are determined by its closest center. Since it is in the batch mode, the centers are updated when all the patterns are assigned to a cluster. This procedure are repeated until the center locations become still.

The code of the algorithm is given below:

```
1 def kMeans_batchMode(clusteringData, num_cluster
2   =5):
3   patterns = np.array(clusteringData[['x', 'y']],
4   dtype=np.float)
5   #class_data = np.array(clusteringData['class
6  '], dtype=np.int)
7   P = np.size(patterns, 0) # pattern size
8   idx = np.random.randint(P, size=num_cluster)
9   W = patterns[idx,:]
10  z_old = np.inf
11  while True:
12    b = np.zeros((P, num_cluster))
13    for n in range(P):
14      idx = np.argmin(np.linalg.norm(patterns[
15      n]-W, axis=1))
16      b[n, idx] = 1
```

```

14     z=0
15     for n in range(P):
16         for i in range(num_cluster):
17             if b[n,i]==1:
18                 z = z + np.linalg.norm(patterns[n
19                                     ]-W[i])**2
20
21         for i in range(num_cluster):
22             if np.sum(b[:,i],axis=0) == 0:
23                 print('Error: The random initial
24                       centers dont converge to given
25                       the number of clusters')
26                 return np.inf,np.nan,np.nan
27             else:
28                 W[i,:] = np.sum(patterns[b[:,i]==1],
29                               axis=0)/np.sum(b[:,i],axis=0)
30
31         if z_old<=z:
32             break
33         z_old=z
34     return z,b,W

```

In order to obtain the better results, the algorithm is run 10.000 times. And the clustering is done with 5, 10, and 15 many clusters. So, the code to obtain the results are shown below:

```

1 # In[:
2 error_min=np.inf
3 for n in range(10000):
4     error,b,W = kMeans_batchMode(data,5)
5     if error<error_min:
6         error_min = error
7         b_min = b
8         W_min = W
9     print(n)
10 plotClusteringGraph(data, W_min, b_min, "
11     Clustering in batch mode with 5 centers", "
12     part1a_N5")
13 # In[:
14 error_min=np.inf
15 for n in range(10000):
16     error,b,W = kMeans_batchMode(data,10)
17     if error<error_min:
18         error_min = error
19         b_min = b
20         W_min = W
21     print(n)
22 plotClusteringGraph(data, W_min, b_min, "
23     Clustering in batch mode with 10 centers", "
24     part1a_N10")
25 # In[:
26 error_min=np.inf
27 for n in range(10000):
28     error,b,W = kMeans_batchMode(data,15)
29     if error<error_min:
30         error_min = error
31         b_min = b
32         W_min = W
33     print(n)
34 plotClusteringGraph(data, W_min, b_min, "
35     Clustering in batch mode with 15 centers", "
36     part1a_N15")

```

The resulting centers and clustered data with 5, 10, and 15 many clusters are shown in the Figure 2, 3, and 4, respectively. In the figures, the circles are the given data and the stars are the found centers of the clusters.

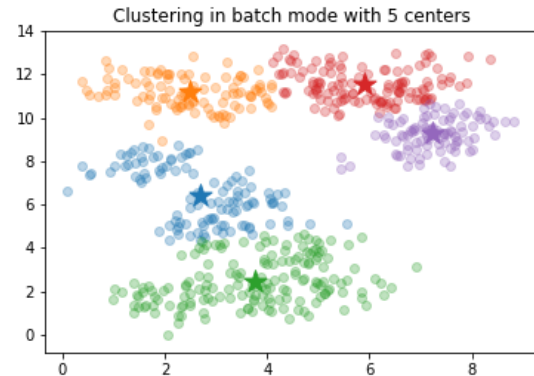


Fig. 2. The clustered data and centers with 5 clusters in batch mode

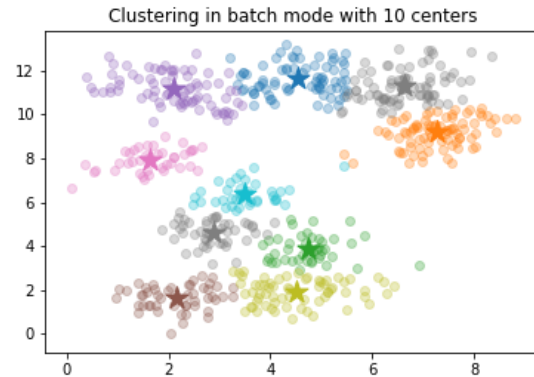


Fig. 3. The clustered data and centers with 10 clusters in batch mode

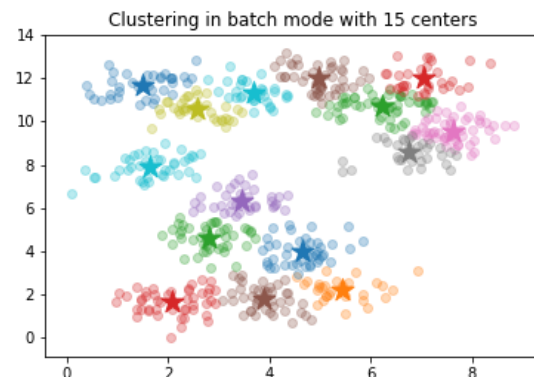


Fig. 4. The clustered data and centers with 15 clusters in batch mode

B. K-mean clustering with on-line mode:

In this method, the initial centers are also selected randomly and the members of the clusters are determined by its closest center. Since it is in the on-line mode, the centers are updated when a pattern is assigned to a cluster. This procedure are repeated until the center locations become still.

The code of the algorithm is given below:

```
1 def kMeans_onlineMode(clusteringData, num_cluster
   =5):
2     patterns = np.array(clusteringData[['x','y']],
   dtype=np.float)
3     #class_data = np.array(clusteringData['class
   '], dtype=np.int)
4     P = np.size(patterns, 0) # pattern size
5     idx = np.random.randint(P, size=num_cluster)
6     W = patterns[idx,:]
7     z_old = np.inf
8     while True:
9         b = np.zeros((P,num_cluster))
10        for n in range(P):
11            idx = np.argmin(np.linalg.norm(patterns[
   n]-W,axis=1))
12            b[n,idx] = 1
13        z=0
14        for n in range(P):
15            for i in range(num_cluster):
16                if b[n,i]==1:
17                    z = z + np.linalg.norm(patterns[n
   ]-W[i])**2
18            for i in range(num_cluster):
19                if np.sum(b[:,i],axis=0) == 0:
20                    print('Error: The random initial
   centers dont converge to given
   the number of clusters')
21                    return np.inf,np.nan,np.nan
22            else:
23                W[i,:] = np.sum(patterns[b[:,i
   ]==1],axis=0)/np.sum(b[:,i],
   axis=0)
24
25        if z_old<=z:
26            break
27        z_old=z
28    return z,b,W
```

The clustering is done with 5, 10, and 15 many clusters. So, the code to obtain the results are shown below:

```
1 # In[:
2 error,b,W = kMeans_onlineMode(data,5)
3 plotClusteringGraph(data, W, b, "Clustering in
   online mode with 5 centers", "part1b_N5")
4
5 # In[:
6 error,b,W = kMeans_onlineMode(data,10)
7 plotClusteringGraph(data, W, b, "Clustering in
   online mode with 10 centers", "part1b_N10")
8
9 # In[:
10 error,b,W = kMeans_onlineMode(data,15)
11 plotClusteringGraph(data, W, b, "Clustering in
   online mode with 15 centers", "part1b_N15")
```

The resulting centers and clustered data with 5, 10, and 15 many clusters are shown in the Figure 5, 6, and 7, respectively.

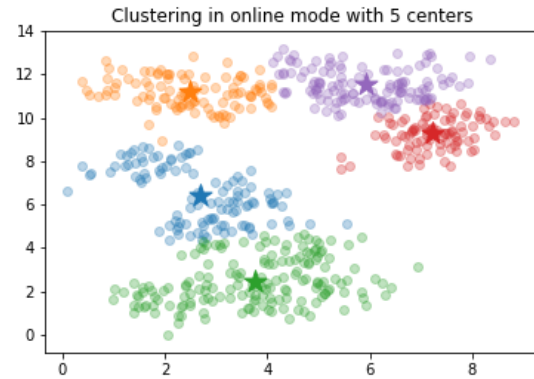


Fig. 5. The clustered data and centers with 5 clusters in on-line mode

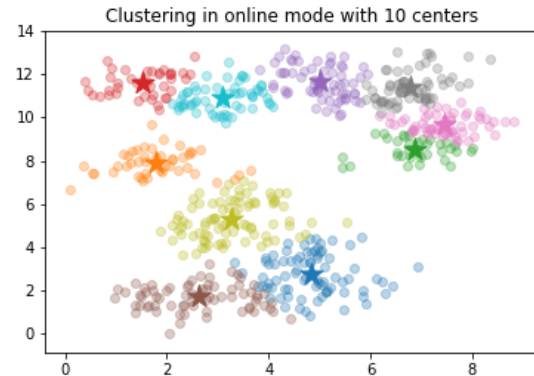


Fig. 6. The clustered data and centers with 10 clusters in on-line mode

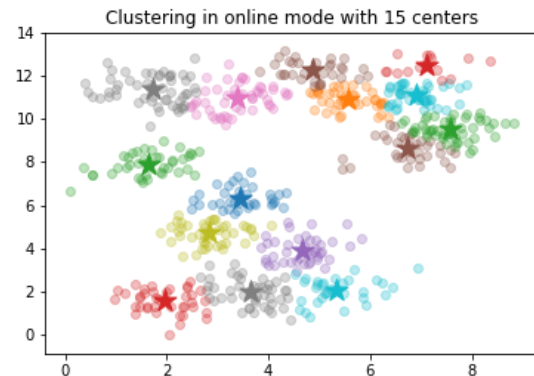


Fig. 7. The clustered data and centers with 15 clusters in on-line mode

C. SOM:

In this method, the initial centers are also selected randomly. The Gaussian kernel with initial $\sigma = I/10$, updated by multiplying 0.7 at each iteration, is used as a neighbor function. And the initial step length used in updating the centers is selected as $\alpha = 0.7$, updated by multiplying 0.7 at each iteration. This procedure are repeated until the center locations become still, in this problem 20 iterations are enough to obtain still centers.

The code of the algorithm is given below:

```
1 def SOM(somData, num_neurons=5, alpha=0.7, sigma
2     =1, beta1=0.7, beta2=0.7):
3     patterns = np.array(somData[['x','y']], dtype=
4         np.float)
5     #class_data = np.array(somData['class'], dtype
6         =np.int)
7     t = 0
8     P = np.size(patterns, 0) # pattern size
9     idx = np.random.randint(P, size=num_neurons)
10    W = patterns[idx,:]
11    while True:
12        np.random.shuffle(patterns)
13        for n in range(P):
14            idx = np.argmin(np.linalg.norm(patterns[
15                n]-W,axis=1))
16            for i in range(num_neurons):
17                neig_func = np.exp(-(np.linalg.norm(W
18                    [i,:]-W[idx,:])/sigma)**2)
19                delta_w = alpha*neig_func*(patterns[n
20                    ,:]-W[i,:])
21                W[i,:] = W[i,:] + delta_w
22            #print(W)
23            if t==20:
24                b = np.zeros((P,num_neurons))
25                for n in range(P):
26                    idx = np.argmin(np.linalg.norm(
27                        patterns[n]-W,axis=1))
28                    b[n,idx] = 1
29                break
30            sigma = beta1*sigma
31            alpha = beta2*alpha
32            t = t + 1
33    return patterns,b,W
```

The clustering is done with 5, 10, and 15 many clusters. So, the code to obtain the results are shown below:

```
1 # In[:]:
2 shuffled_data,b,W = SOM(data,5, sigma=5/10)
3 plotClusteringGraph(shuffled_data, W, b, "Self
4     organizing map with 5 neurons", "part1c_N5")
5
6 # In[:]:
7 shuffled_data,b,W = SOM(data,10,sigma=10/10)
8 plotClusteringGraph(shuffled_data, W, b, "Self
9     organizing map with 10 neurons", "part1c_N10"
10 )
11
12 # In[:]:
13 shuffled_data,b,W = SOM(data,15,sigma=15/10)
```

```
11 plotClusteringGraph(shuffled_data, W, b, "Self
    organizing map with 15 neurons", "part1c_N15"
    )
```

The resulting centers and clustered data with 5, 10, and 15 many clusters are shown in the Figure 8, 9, and 10, respectively.

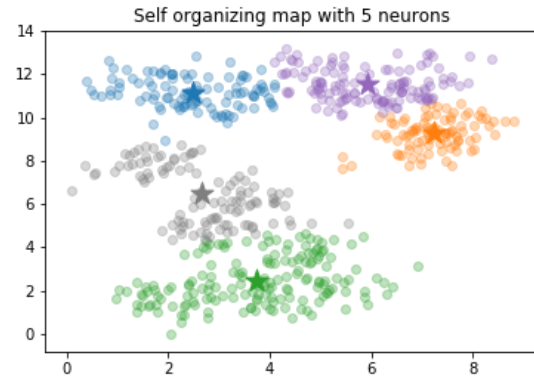


Fig. 8. The clustered data and centers with 5 neurons using SOM

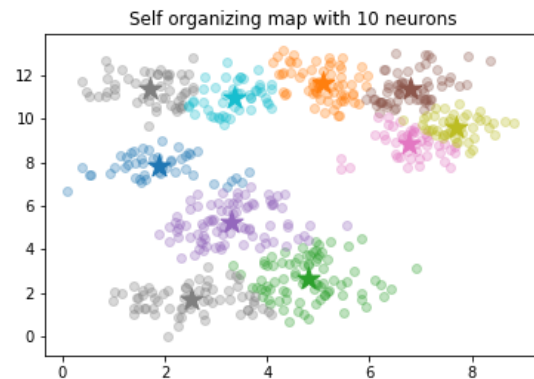


Fig. 9. The clustered data and centers with 10 neurons using SOM

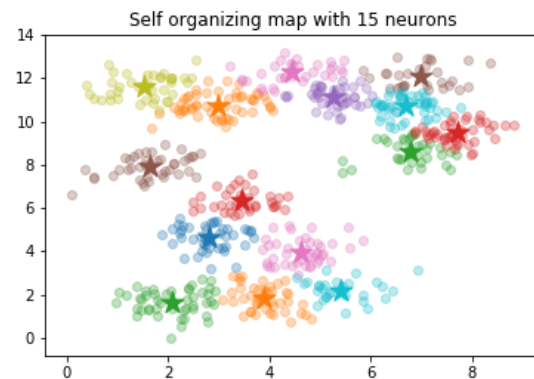


Fig. 10. The clustered data and centers with 15 neurons using SOM

II. SOM FOR THE EUCLIDEAN TSP

In this problem, given the coordinates of the cities, the tour for a salesman is tried to find with a minimum total destination.

The problem is solved exactly using *cplex*. The optimum tour is shown in the Figure 11, and the total destination is 3904.21 km.

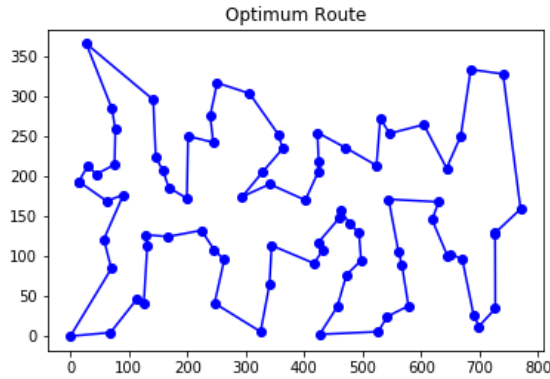


Fig. 11. The optimum route found by cplex solver

In the SOM algorithm, 2 different neighborhood functions are implemented: Gaussian Kernel and a neighborhood function defined on the elastic band. However, the Gaussian Kernel neighborhood function is not appropriate for TSP since it doesn't take into account the ordering of the neurons yet the other one considers it and gives a path between the actual cities.

The initial centers are also selected randomly. And the parameters are selected as following: The initial σ of the neighborhood functions is $1/10$, updated by multiplying 0.8 at each iteration. And the initial step length used in updating the centers is selected as $\alpha = 1$, updated by multiplying 0.8 at each iteration. This procedure are repeated until the neurons converges to the city locations, in this problem 100 iterations are enough to obtain the convergence.

The code of the algorithm is given below:

```
1 def SOM_TSP(somData, num_neurons=81, neigFunc=0,
2   alpha=1, sigma=1, betal=0.8, beta2=0.8):
3   patterns = np.array(somData[['x','y']], dtype=
4     np.float)
5   patterns = np.c_[patterns,np.linspace(1,81,81)
6     ]
7   t = 0
8   P = np.size(patterns, 0) # pattern size
```

```
6   W = np.array([[np.random.uniform(0.0, patterns
7     [:,0].max()), np.random.uniform(0.0,
8     patterns[:,1].max()),i] for i in range(
9     num_neurons)])
10  while True:
11    np.random.shuffle(patterns)
12    for n in range(P):
13      idx = np.argmin(np.linalg.norm(patterns[
14        n,[0,1]]-W[:,[0,1]],axis=1))
15      for i in range(num_neurons):
16        if neigFunc==0:
17          neig_func = np.exp(-(np.linalg.
18            norm(W[i,[0,1]]-W[idx,[0,1]])/
19            sigma)**2)
20        else:
21          d = np.min([np.abs(i-idx),
22            num_neurons-np.abs(i-idx)])
23          neig_func = np.exp(-(d/sigma)**2)
24      print(t)
25      if t==100:
26        b = np.zeros((P,num_neurons))
27        ordering = np.zeros((P,1))
28        for n in range(P):
29          idx = np.argmin(np.linalg.norm(
30            patterns[n,[0,1]]-W[:,[0,1]],axis
31            =1))
32          b[n,idx] = 1
33          ordering[n]=idx
34        patterns = np.c_[patterns,ordering]
35        patterns = patterns[patterns[:,3].
36          argsort()]
37        patterns = np.vstack([patterns, patterns
38          [0,:]])
39        break
40      sigma = betal*sigma
41      alpha = beta2*alpha
42      t = t + 1
43  return patterns,b,W
```

The results are obtained by the following code:

```
1 # In[:
2 num_city = data_tsp.index[:].size
3 ordered_data,b,W = SOM_TSP(data_tsp,num_city*5,
4   neigFunc=0, sigma=1)
5 plotSOM_TSPGraph(ordered_data, W, b, "Self
6   organizing map with 81*5 neurons - GK", "
7   part2a")
8 dist = cal_dist(ordered_data)
9 # In[:
10 ordered_data,b,W = SOM_TSP(data_tsp,num_city*5,
11   neigFunc=1, sigma=5*num_city/10)
12 plotSOM_TSPGraph(ordered_data, W, b, "Self
13   organizing map with 81*5 neurons - EB", "
14   part2b_51")
15 dist = cal_dist(ordered_data)
```

In order to achieve the convergence, the number of neurons are selected as 5 times of the number of cities.

The plots of the city locations and the final tours which are obtained using the Gaussian Kernel neighborhood function and the neighborhood function on the elastic band are shown in the Figure 12, and 13. The blue circles are the city locations, the lines are the found tour, and the red circles

are the neuron locations. The length of the total destination is 24005 km with Gaussian Kernel (GK), 3986.64 km with the neighborhood function on the elastic band (EB).

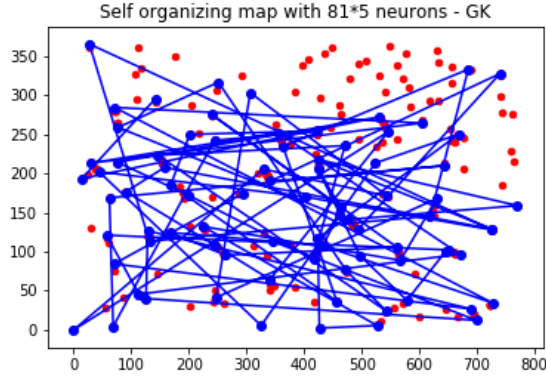


Fig. 12. The city locations and the final tour of the GK

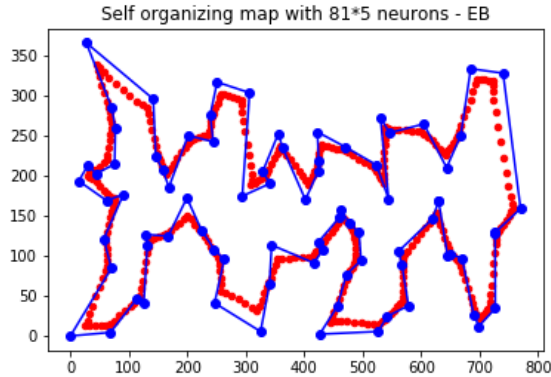


Fig. 13. The city locations and the final tour of the EB

So, it can be seen from the figures and the found destination length that the SOM method can find a solution very close (In optimum solution, the length was 3904.21; in the SOM, it is 3986) to the optimum solution to the travelling salesman problem.

III. APPENDIX

The complete source code:

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Tue Dec 31 11:22:56 2019
4
5 @author: SEFA
6 """
```

```
7 # In[]:
8 import pandas as pd
9 import numpy as np
10 import matplotlib.pyplot as plt
11
12
13 data = pd.read_csv('IE440Final19ClusteringData.
14 txt', sep='\t', header=None, names=['x', 'y',
15 'class']);
16 data = data.drop(data.index[0])
17
18 data_tsp = pd.read_csv('IE440Final19ETSPData.txt'
19 , sep=',', header=None, names=['City', 'x', '
20 y']);
21 data_tsp = data_tsp.drop(data_tsp.index[0])
22
23 # In[]:
24 def plotClusteringGraph(data, w_star, b_star,
25 title, name="graph"):
26 if isinstance(data, np.ndarray):
27 patterns = data
28 else:
29 patterns = np.array(data[['x','y']], dtype=
30 np.float)
31 N = np.size(b_star,1)
32 colors = ['tab:blue', 'tab:orange', 'tab:green
33 ', 'tab:red', 'tab:purple', 'tab:brown', '
34 tab:pink', 'tab:gray', 'tab:olive', 'tab:
35 cyan', 'tab:brown', 'tab:orange', 'tab:
36 green', 'tab:red', 'tab:cyan']
37 plt.figure()
38 for n in range(N):
39 plt.scatter(w_star[n,0],w_star[n,1], marker
40 ="*", s=250, label=n+1,color=colors[n])
41 plt.scatter(patterns[b_star[:,n]==1,0],
42 patterns[b_star[:,n]==1,1],color=colors
43 [n],alpha=0.3)
44 plt.title(title)
45 plt.savefig("{0}.png".format(name))
46
47 # ### Part 1:
48 # In[]:
49 def kMeans_batchMode(clusteringData, num_cluster
50 =5):
51 patterns = np.array(clusteringData[['x','y']],
52 dtype=np.float)
53 #class_data = np.array(clusteringData['class
54 '], dtype=np.int)
55 P = np.size(patterns, 0) # pattern size
56 idx = np.random.randint(P, size=num_cluster)
57 W = patterns[idx,:]
58 z_old = np.inf
59 while True:
60 b = np.zeros((P,num_cluster))
61 for n in range(P):
62 idx = np.argmin(np.linalg.norm(patterns[
63 n]-W,axis=1))
64 b[n,idx] = 1
65
66 z=0
67 for n in range(P):
68 for i in range(num_cluster):
69 if b[n,i]==1:
70 z = z + np.linalg.norm(patterns[n
71 ]-W[i])**2
72
73 for i in range(num_cluster):
74 if np.sum(b[:,i],axis=0) == 0:
```

```

57         print('Error: The random initial
           centers dont converge to given
           the number of clusters')
58         return np.inf,np.nan,np.nan
59     else:
60         W[i,:] = np.sum(patterns[b[:,i]==1],
           axis=0)/np.sum(b[:,i],axis=0)
61
62         if z_old<=z:
63             break
64         z_old=z
65         return z,b,W
66 # In[]:
67 data_array = np.array(data[['x','y', 'class']],
           dtype=np.float)
68 colors = ['tab:blue', 'tab:orange', 'tab:pink', '
           tab:gray', 'tab:purple', 'tab:brown', 'tab:
           pink', 'tab:gray', 'tab:olive', 'tab:red', '
           tab:brown', 'tab:orange', 'tab:green', 'tab:
           red', 'tab:cyan']
69 for n in range(15):
70     plt.scatter(data_array[data_array[:,2]==n
           +1,0], data_array[data_array[:,2]==n+1,1],
           c=colors[n],label=n)
71 #plt.legend()
72 plt.title("Original Classes")
73 plt.savefig("{0}.png".format("part1_original"))
74 # In[]:
75 error_min=np.inf
76 for n in range(10000):
77     error,b,W = kMeans_batchMode(data,5)
78     if error<error_min:
79         error_min = error
80         b_min = b
81         W_min = W
82     print(n)
83 plotClusteringGraph(data, W_min, b_min, "
           Clustering in batch mode with 5 centers", "
           part1a_N5")
84
85 # In[]:
86 error_min=np.inf
87 for n in range(10000):
88     error,b,W = kMeans_batchMode(data,10)
89     if error<error_min:
90         error_min = error
91         b_min = b
92         W_min = W
93     print(n)
94 plotClusteringGraph(data, W_min, b_min, "
           Clustering in batch mode with 10 centers", "
           part1a_N10")
95
96 # In[]:
97 error_min=np.inf
98 for n in range(10000):
99     error,b,W = kMeans_batchMode(data,15)
100     if error<error_min:
101         error_min = error
102         b_min = b
103         W_min = W
104     print(n)
105 plotClusteringGraph(data, W_min, b_min, "
           Clustering in batch mode with 15 centers", "
           part1a_N15")
106
107 # In[]:
108 def kMeans_onlineMode(clusteringData, num_cluster
           =5):
109     patterns = np.array(clusteringData[['x','y']],
           dtype=np.float)
110     #class_data = np.array(clusteringData['class
           '], dtype=np.int)
111     P = np.size(patterns, 0) # pattern size
112     idx = np.random.randint(P, size=num_cluster)
113     W = patterns[idx,:]
114     z_old = np.inf
115     while True:
116         b = np.zeros((P,num_cluster))
117         for n in range(P):
118             idx = np.argmin(np.linalg.norm(patterns[
           n]-W,axis=1))
119             b[n,idx] = 1
120         z=0
121         for n in range(P):
122             for i in range(num_cluster):
123                 if b[n,i]==1:
124                     z = z + np.linalg.norm(patterns[n
           ]-W[i])**2
125             for i in range(num_cluster):
126                 if np.sum(b[:,i],axis=0) == 0:
127                     print('Error: The random initial
           centers dont converge to given
           the number of clusters')
128                     return np.inf,np.nan,np.nan
129             else:
130                 W[i,:] = np.sum(patterns[b[:,i]
           ==1],axis=0)/np.sum(b[:,i],
           axis=0)
131
132         if z_old<=z:
133             break
134         z_old=z
135         return z,b,W
136
137 # In[]:
138 error,b,W = kMeans_onlineMode(data,5)
139 plotClusteringGraph(data, W, b, "Clustering in
           online mode with 5 centers", "part1b_N5")
140
141 # In[]:
142 error,b,W = kMeans_onlineMode(data,10)
143 plotClusteringGraph(data, W, b, "Clustering in
           online mode with 10 centers", "part1b_N10")
144
145 # In[]:
146 error,b,W = kMeans_onlineMode(data,15)
147 plotClusteringGraph(data, W, b, "Clustering in
           online mode with 15 centers", "part1b_N15")
148
149 # In[]:
150 def SOM(somData, num_neurons=5, alpha=0.7, sigma
           =1, beta1=0.7, beta2=0.7):
151     patterns = np.array(somData[['x','y']], dtype=
           np.float)
152     #class_data = np.array(somData['class'], dtype
           =np.int)
153     t = 0
154     P = np.size(patterns, 0) # pattern size
155     idx = np.random.randint(P, size=num_neurons)
156     W = patterns[idx,:]
157     while True:
158         np.random.shuffle(patterns)
159         for n in range(P):
160             idx = np.argmin(np.linalg.norm(patterns[
           n]-W,axis=1))
161             for i in range(num_neurons):

```



```

163     neig_func = np.exp(-(np.linalg.norm(W
164         [i,:]-W[idx,:])/sigma)**2)
165     delta_w = alpha*neig_func*(patterns[n
166         ,:]-W[i,:])
167     W[i,:] = W[i,:] + delta_w
168     #print(W)
169     if t==20:
170         b = np.zeros((P,num_neurons))
171         for n in range(P):
172             idx = np.argmin(np.linalg.norm(
173                 patterns[n]-W,axis=1))
174             b[n,idx] = 1
175         break
176     sigma = beta1*sigma
177     alpha = beta2*alpha
178     t = t + 1
179     return patterns,b,W
180 # In[:]:
181 shuffled_data,b,W = SOM(data,5, sigma=5/10)
182 plotClusteringGraph(shuffled_data, W, b, "Self
183     organizing map with 5 neurons", "part1c_N5")
184 # In[:]:
185 shuffled_data,b,W = SOM(data,10,sigma=10/10)
186 plotClusteringGraph(shuffled_data, W, b, "Self
187     organizing map with 10 neurons", "part1c_N10")
188 # In[:]:
189 shuffled_data,b,W = SOM(data,15,sigma=15/10)
190 plotClusteringGraph(shuffled_data, W, b, "Self
191     organizing map with 15 neurons", "part1c_N15")
192 # In[:]:
193 def cal_dist(data):
194     data = data[:,[0,1]]
195     N = np.size(data,0)
196     total=0
197     for n in range(N-1):
198         total += np.linalg.norm(data[n+1,:]-data[n
199             ,:])
200     total += np.linalg.norm(data[N-1,:]-data[0,:])
201     return total
202 # In[:]:
203 def plotSOM_TSPGraph(data, w_star, b_star, title,
204     name="graph"):
205     patterns = data[:,[0,1]]
206     N = np.size(b_star,1)
207     plt.figure()
208     for n in range(N):
209         plt.scatter(w_star[n,0],w_star[n,1], s=20,
210             label=n+1,color="red")
211     plt.plot(patterns[:,0], patterns[:,1], 'bo-')
212     plt.title(title)
213     plt.savefig("{0}.png".format(name))
214 # ### Part 2:
215 # In[:]:
216 def SOM_TSP(somData, num_neurons=81, neigFunc=0,
217     alpha=1, sigma=1, beta1=0.8, beta2=0.8):
218     patterns = np.array(somData[['x','y']], dtype=
219         np.float)
220     patterns = np.c_[patterns,np.linspace(1,81,81)
221         ]
222     t = 0
223     P = np.size(patterns, 0) # pattern size
224     W = np.array([(np.random.uniform(0.0, patterns
225        [:,0].max()), np.random.uniform(0.0,
226         patterns[:,1].max()),i] for i in range(
227         num_neurons)])
228     while True:
229         np.random.shuffle(patterns)
230         for n in range(P):
231             idx = np.argmin(np.linalg.norm(patterns[
232                 n,[0,1]]-W[:,[0,1]],axis=1))
233             for i in range(num_neurons):
234                 if neigFunc==0:
235                     neig_func = np.exp(-(np.linalg.
236                         norm(W[i,[0,1]]-W[idx,[0,1]])/
237                         sigma)**2)
238                 else:
239                     d = np.min([np.abs(i-idx),
240                         num_neurons-np.abs(i-idx)])
241                     neig_func = np.exp(-(d/sigma)**2)
242             print(t)
243             if t==100:
244                 b = np.zeros((P,num_neurons))
245                 ordering = np.zeros((P,1))
246                 for n in range(P):
247                     idx = np.argmin(np.linalg.norm(
248                         patterns[n,[0,1]]-W[:,[0,1]],axis
249                         =1))
250                     b[n,idx] = 1
251                     ordering[n]=idx
252                 patterns = np.c_[patterns,ordering]
253                 patterns = patterns[patterns[:,3].
254                     argsort()]
255                 patterns = np.vstack([patterns, patterns
256                     [0,:]])
257                 break
258             sigma = beta1*sigma
259             alpha = beta2*alpha
260             t = t + 1
261         return patterns,b,W
262 # In[:]:
263 optimum_route = np.array
264     ([1,6,24,58,37,67,69,46,51,23,33,44,63,34,
265     43,13,81,75,27,19,71,35,15,8,66,65,52,79,
266     10,40,9,45,73,49,28,22,38,36,41,56,78,11,
267     12,4,72,59,21,25,68,18,60,77,3,17,54,14,
268     31,20,57,16,32,39,26,30,76,29,55,53,2,7,
269     48,42,70,5,50,61,47,62,80,74,64])
270 optimum_order = np.array(data_tsp[['x','y']],
271     dtype=np.float)
272 optimum_order = optimum_order[optimum_route-1,:]
273 optimum_order = np.vstack([optimum_order,
274     optimum_order[0,:]])
275 plt.plot(optimum_order[:,0], optimum_order[:,1], '
276     bo-')
277 plt.title("Optimum Route")
278 plt.savefig("{0}.png".format("part2_optimum"))
279 dist = cal_dist(optimum_order)
280 # In[:]:
281 num_city = data_tsp.index[:].size
282 ordered_data,b,W = SOM_TSP(data_tsp,num_city*5,
283     neigFunc=0, alpha=1, sigma=0.5, beta1=0.7,
284     beta2=0.8)
285 plotSOM_TSPGraph(ordered_data, W, b, "Self
286     organizing map with 81*5 neurons - GK", "
287     part2a")
288 dist = cal_dist(ordered_data)
289 # In[:]:
290 ordered_data,b,W = SOM_TSP(data_tsp,num_city*5,
291     neigFunc=1, sigma=5*num_city/10)

```

```
267 | plotSOM_TSPGraph(ordered_data, W, b, "Self  
      organizing map with 81*5 neurons - EB", "  
      part2b_5")  
268 | dist = cal_dist(ordered_data)
```