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

Sefa Kayraklık
2015401066

Final Project

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:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


data = pd.read_csv('IE440Final19ClusteringData.txt', sep='\t', header=None, names=['x', 'y', 'class']);
data = data.drop(data.index[0])

data_tsp = pd.read_csv('IE440Final19ETSPData.txt', sep=',', header=None, names=['City', 'x', 'y']);
data_tsp = data_tsp.drop(data_tsp.index[0])
```

Some useful functions for plot construction:

```
def plotClusteringGraph(data, w_star, b_star, title, name="graph"):
    if isinstance(data, np.ndarray):
        patterns = data
    else:
        patterns = np.array(data[['x','y']], dtype=np.float)
    N = np.size(b_star,1)
    colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown', 'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan','tab:brown', 'tab:orange', 'tab:green', 'tab:red', 'tab:cyan']
    plt.figure()
    for n in range(N):
        plt.scatter(w_star[n,0],w_star[n,1], marker="*", s=250, label=n+1,color=colors[n])
        plt.scatter(patterns[b_star[:,n]==1,0], patterns[b_star[:,n]==1,1],color=colors[n],alpha=0.3)
    plt.title(title)
    plt.savefig("{0}.png".format(name))

def cal_dist(data):
    data = data[:,[0,1]]
    N = np.size(data,0)
    total=0
    for n in range(N-1):
        total += np.linalg.norm(data[n+1,:]-data[n,:])
    total += np.linalg.norm(data[N-1,:]-data[0,:])
    return total

def plotSOM_TSPGraph(data, w_star, b_star, title, name="graph"):
    patterns = data[:,[0,1]]
    N = np.size(b_star,1)
    plt.figure()
    for n in range(N):
        plt.scatter(w_star[n,0],w_star[n,1], s=20, label=n+1,color="red")
    plt.plot(patterns[:,0], patterns[:,1],'bo-')
    plt.title(title)
    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.

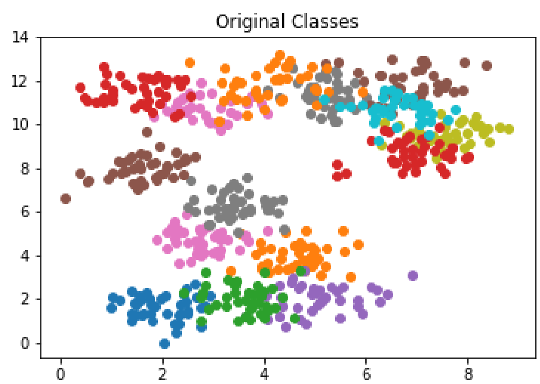

Fig. 1. The original data and its clusters

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:

```
def kMeans_batchMode(clusteringData, num_cluster=5):
    patterns = np.array(clusteringData[['x','y']], dtype=np.float)
    #class_data = np.array(clusteringData['class'], dtype=np.int)
    P = np.size(patterns, 0) # pattern size
    idx = np.random.randint(P, size=num_cluster)
    W = patterns[idx,:]
    z_old = np.inf
    while True:
        b = np.zeros((P,num_cluster))
        for n in range(P):
            idx = np.argmin(np.linalg.norm(patterns[n]-W,axis=1))
            b[n,idx] = 1

```

```
        z=0
        for n in range(P):
            for i in range(num_cluster):
                if b[n,i]==1:
                    z = z + np.linalg.norm(patterns[n
                        ]-W[i])**2

        for i in range(num_cluster):
            if np.sum(b[:,i],axis=0) == 0:
                print('Error: The random initial
                    centers dont converge to given
                    the number of clusters')
                return np.inf,np.nan,np.nan
            else:
                W[i,:] = np.sum(patterns[b[:,i]==1],
                    axis=0)/np.sum(b[:,i],axis=0)

        if z_old<=z:
            break
        z_old=z
    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:

```
# In[]:
error_min=np.inf
for n in range(10000):
    error,b,W = kMeans_batchMode(data,5)
    if error<error_min:
        error_min = error
        b_min = b
        W_min = W
    print(n)
plotClusteringGraph(data, W_min, b_min, "
    Clustering in batch mode with 5 centers", "
    part1a_N5")

# In[]:
error_min=np.inf
for n in range(10000):
    error,b,W = kMeans_batchMode(data,10)
    if error<error_min:
        error_min = error
        b_min = b
        W_min = W
    print(n)
plotClusteringGraph(data, W_min, b_min, "
    Clustering in batch mode with 10 centers", "
    part1a_N10")

# In[]:
error_min=np.inf
for n in range(10000):
    error,b,W = kMeans_batchMode(data,15)
    if error<error_min:
        error_min = error
        b_min = b
        W_min = W
    print(n)
plotClusteringGraph(data, W_min, b_min, "
    Clustering in batch mode with 15 centers", "
    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.

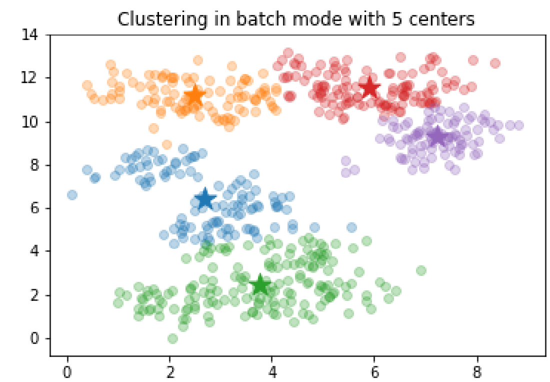

Fig. 2. The clustered data and centers with 5 clusters in batch mode

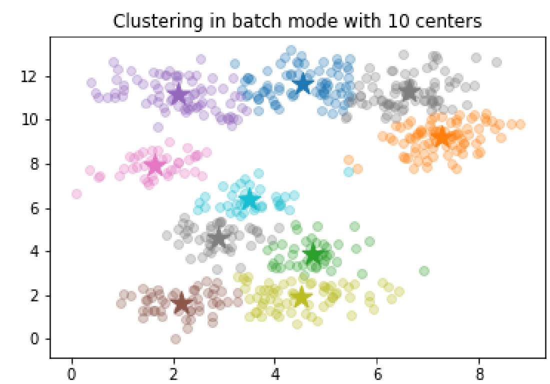

Fig. 3. The clustered data and centers with 10 clusters in batch mode

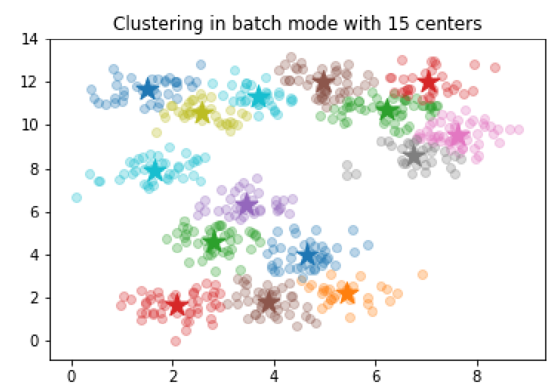

Fig. 4. The clustered data and centers with 15 clusters in batch mode

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:

```python
def kMeans_onlineMode(clusteringData, num_cluster=5):
    patterns = np.array(clusteringData[['x','y']], dtype=np.float)
    #class_data = np.array(clusteringData['class'], dtype=np.int)
    P = np.size(patterns, 0) # pattern size
    idx = np.random.randint(P, size=num_cluster)
    W = patterns[idx,:]
    z_old = np.inf
    while True:
        b = np.zeros((P,num_cluster))
        for n in range(P):
            idx = np.argmin(np.linalg.norm(patterns[n]-W,axis=1))
            b[n,idx] = 1
        z=0
        for n in range(P):
            for i in range(num_cluster):
                if b[n,i]==1:
                    z = z + np.linalg.norm(patterns[n]-W[i])**2
            for i in range(num_cluster):
                if np.sum(b[:,i],axis=0) == 0:
                    print('Error: The random initial centers dont converge to given the number of clusters')
                    return np.inf,np.nan,np.nan
                else:
                    W[i,:] = np.sum(patterns[b[:,i]==1],axis=0)/np.sum(b[:,i],axis=0)


        if z_old<=z:
            break
        z_old=z
    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:

```python
# In[]:
error,b,W = kMeans_onlineMode(data,5)
plotClusteringGraph(data, W, b, "Clustering in online mode with 5 centers", "part1b_N5")

# In[]:
error,b,W = kMeans_onlineMode(data,10)
plotClusteringGraph(data, W, b, "Clustering in online mode with 10 centers", "part1b_N10")

# In[]:
error,b,W = kMeans_onlineMode(data,15)
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.

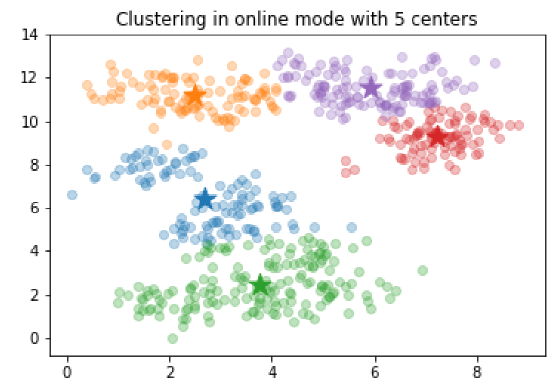

Fig. 5. The clustered data and centers with 5 clusters in on-line mode

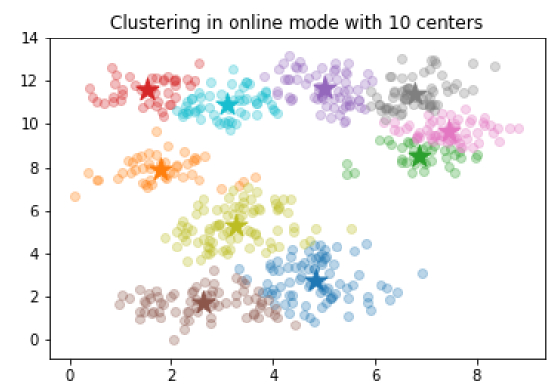

Fig. 6. The clustered data and centers with 10 clusters in on-line mode

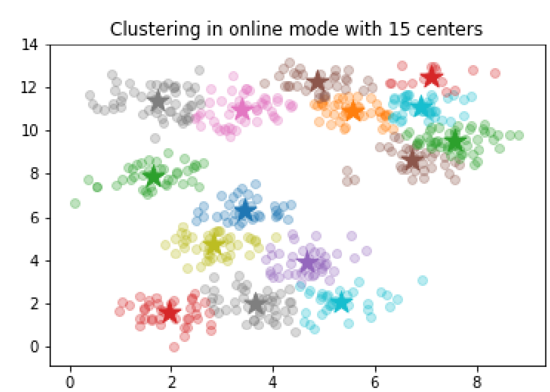

Fig. 7. The clustered data and centers with 15 clusters in on-line mode

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:

```
def SOM(somData, num_neurons=5, alpha=0.7, sigma
    =1, beta1=0.7, beta2=0.7):
    patterns = np.array(somData[['x','y']], dtype=
        np.float)
    #class_data = np.array(somData['class'], dtype
        =np.int)
    t = 0
    P = np.size(patterns, 0) # pattern size
    idx = np.random.randint(P, size=num_neurons)
    W = patterns[idx,:]
    while True:
        np.random.shuffle(patterns)
        for n in range(P):
            idx = np.argmin(np.linalg.norm(patterns[
                n]-W,axis=1))
            for i in range(num_neurons):
                neig_func = np.exp(-(np.linalg.norm(W
                    [i,:]-W[idx,:])/sigma)**2)
                delta_w = alpha*neig_func*(patterns[n
                    ,:]-W[i,:])
                W[i,:] = W[i,:] + delta_w
        #print(W)
        if t==20:
            b = np.zeros((P,num_neurons))
            for n in range(P):
                idx = np.argmin(np.linalg.norm(
                    patterns[n]-W,axis=1))
                b[n,idx] = 1
            break
        sigma = beta1*sigma
        alpha = beta2*alpha
        t = t + 1
    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:

```
# In[]:
shuffled_data,b,W = SOM(data,5, sigma=5/10)
plotClusteringGraph(shuffled_data, W, b, "Self
    organizing map with 5 neurons", "part1c_N5")

# In[]:
shuffled_data,b,W = SOM(data,10,sigma=10/10)
plotClusteringGraph(shuffled_data, W, b, "Self
    organizing map with 10 neurons", "part1c_N10"
    )

# In[]:
shuffled_data,b,W = SOM(data,15,sigma=15/10)
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.

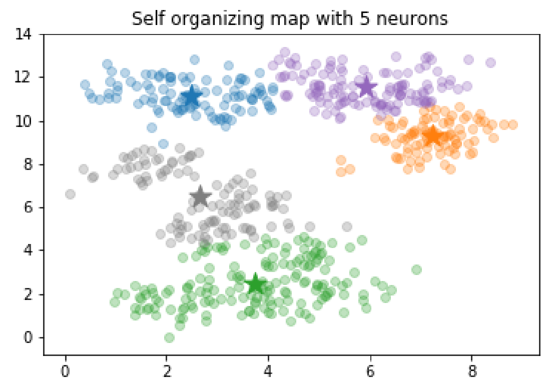

Fig. 8. The clustered data and centers with 5 neurons using SOM

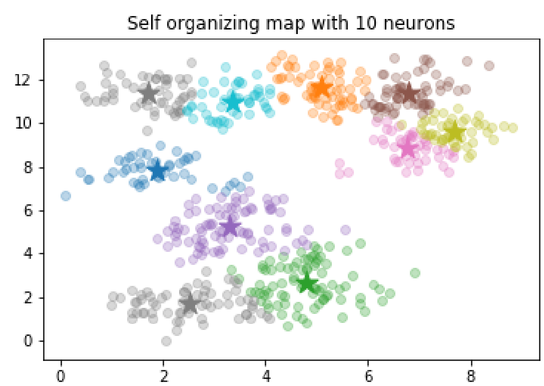

Fig. 9. The clustered data and centers with 10 neurons using SOM

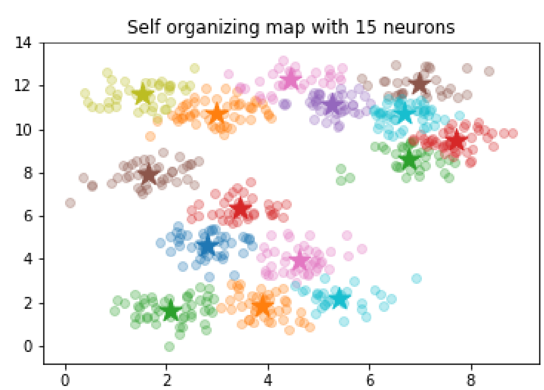

Fig. 10. The clustered data and centers with 15 neurons using SOM

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.

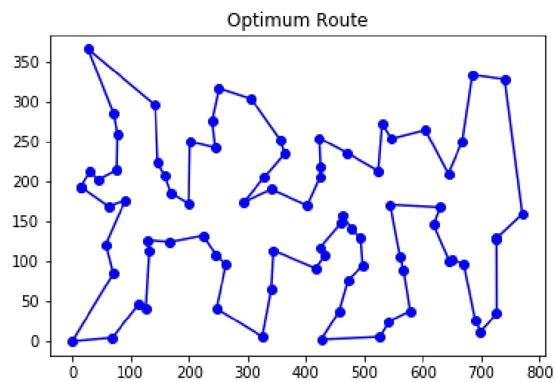

Fig. 11. The optimum route found by cplex solver

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 $I/10$, 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:

```python
def SOM_TSP(somData, num_neurons=81, neigFunc=0, alpha=1, sigma=1, beta1=0.8, beta2=0.8):
    patterns = np.array(somData[['x','y']], dtype=np.float)
    patterns = np.c_[patterns,np.linspace(1,81,81)]
    t = 0
    P = np.size(patterns, 0) # pattern size
    W = np.array([[np.random.uniform(0.0, patterns[:,0].max()), np.random.uniform(0.0, patterns[:,1].max()),i] for i in range(num_neurons)])
    while True:
        np.random.shuffle(patterns)
        for n in range(P):
            idx = np.argmin(np.linalg.norm(patterns[n,[0,1]]-W[:,[0,1]],axis=1))
            for i in range(num_neurons):
                if neigFunc==0:
                    neig_func = np.exp(-(np.linalg.norm(W[i,[0,1]]-W[idx,[0,1]])/sigma)**2)
                else:
                    d = np.min([np.abs(i-idx), num_neurons-np.abs(i-idx)])
                    neig_func = np.exp(-(d/sigma)**2)
        print(t)
        if t==100:
            b = np.zeros((P,num_neurons))
            ordering = np.zeros((P,1))
            for n in range(P):
                idx = np.argmin(np.linalg.norm(patterns[n,[0,1]]-W[:,[0,1]],axis=1))
                b[n,idx] = 1
                ordering[n]=idx
            patterns = np.c_[patterns,ordering]
            patterns = patterns[patterns[:,3].argsort()]
            patterns = np.vstack([patterns, patterns[0,:]])
            break
        sigma = beta1*sigma
        alpha = beta2*alpha
        t = t + 1
    return patterns,b,W
```

The results are obtained by the following code:

```python
# In[]:
num_city = data_tsp.index[:].size
ordered_data,b,W = SOM_TSP(data_tsp,num_city*5, neigFunc=0, sigma=1)
plotSOM_TSPGraph(ordered_data, W, b, "Self organizing map with 81*5 neurons - GK", "part2a")
dist = cal_dist(ordered_data)

# In[]:
ordered_data,b,W = SOM_TSP(data_tsp,num_city*5, neigFunc=1, sigma=5*num_city/10)
plotSOM_TSPGraph(ordered_data, W, b, "Self organizing map with 81*5 neurons - EB", "part2b_51")
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).

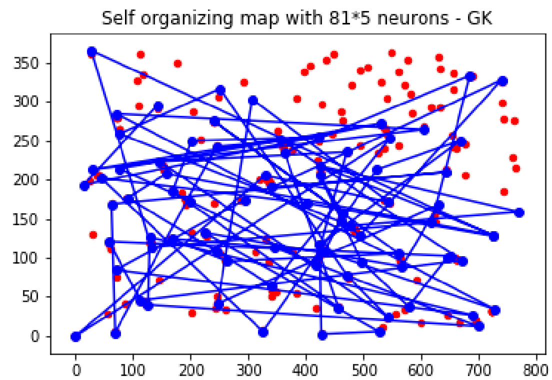

Fig. 12. The city locations and the final tour of the GK

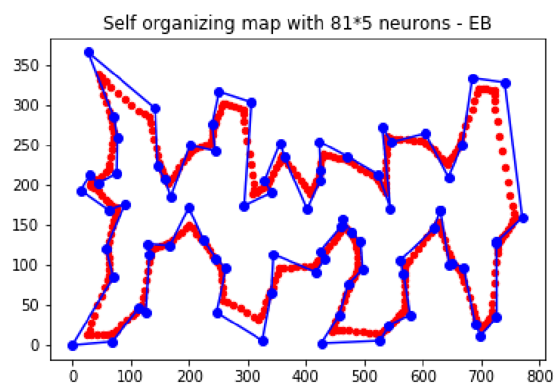

Fig. 13. The city locations and the final tour of the EB

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:

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Dec 31 11:22:56 2019

@author: SEFA
"""
# In[]:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


data = pd.read_csv('IE440Final19ClusteringData.txt', sep='\t', header=None, names=['x', 'y', 'class']);
data = data.drop(data.index[0])

data_tsp = pd.read_csv('IE440Final19ETSPData.txt', sep=',', header=None, names=['City', 'x', 'y']);
data_tsp = data_tsp.drop(data_tsp.index[0])

# In[]:
def plotClusteringGraph(data, w_star, b_star, title, name="graph"):
   if isinstance(data, np.ndarray):
      patterns = data
   else:
      patterns = np.array(data[['x','y']], dtype=np.float)
   N = np.size(b_star,1)
   colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown', 'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan','tab:brown', 'tab:orange', 'tab:green', 'tab:red', 'tab:cyan']
   plt.figure()
   for n in range(N):
      plt.scatter(w_star[n,0],w_star[n,1], marker="*", s=250, label=n+1,color=colors[n])
      plt.scatter(patterns[b_star[:,n]==1,0], patterns[b_star[:,n]==1,1],color=colors[n],alpha=0.3)
   plt.title(title)
   plt.savefig("{0}.png".format(name))

# ### Part 1:
# In[]:
def kMeans_batchMode(clusteringData, num_cluster=5):
   patterns = np.array(clusteringData[['x','y']], dtype=np.float)
   #class_data = np.array(clusteringData['class'], dtype=np.int)
   P = np.size(patterns, 0) # pattern size
   idx = np.random.randint(P, size=num_cluster)
   W = patterns[idx,:]
   z_old = np.inf
   while True:
      b = np.zeros((P,num_cluster))
      for n in range(P):
         idx = np.argmin(np.linalg.norm(patterns[n]-W,axis=1))
         b[n,idx] = 1

      z=0
      for n in range(P):
         for i in range(num_cluster):
            if b[n,i]==1:
               z = z + np.linalg.norm(patterns[n]-W[i])**2

      for i in range(num_cluster):
         if np.sum(b[:,i],axis=0) == 0:
```

```python
                print('Error: The random initial centers dont converge to given the number of clusters')
                return np.inf,np.nan,np.nan
            else:
                W[i,:] = np.sum(patterns[b[:,i]==1],axis=0)/np.sum(b[:,i],axis=0)

        if z_old<=z:
            break
        z_old=z
    return z,b,W
# In[]:
data_array = np.array(data[['x','y', 'class']], dtype=np.float)
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']
for n in range(15):
    plt.scatter(data_array[data_array[:,2]==n+1,0], data_array[data_array[:,2]==n+1,1], c=colors[n],label=n)
#plt.legend()
plt.title("Original Classes")
plt.savefig("{0}.png".format("part1_original"))
# In[]:
error_min=np.inf
for n in range(10000):
    error,b,W = kMeans_batchMode(data,5)
    if error<error_min:
        error_min = error
        b_min = b
        W_min = W
    print(n)
plotClusteringGraph(data, W_min, b_min, "Clustering in batch mode with 5 centers", "part1a_N5")

# In[]:
error_min=np.inf
for n in range(10000):
    error,b,W = kMeans_batchMode(data,10)
    if error<error_min:
        error_min = error
        b_min = b
        W_min = W
    print(n)
plotClusteringGraph(data, W_min, b_min, "Clustering in batch mode with 10 centers", "part1a_N10")

# In[]:
error_min=np.inf
for n in range(10000):
    error,b,W = kMeans_batchMode(data,15)
    if error<error_min:
        error_min = error
        b_min = b
        W_min = W
    print(n)
plotClusteringGraph(data, W_min, b_min, "Clustering in batch mode with 15 centers", "part1a_N15")

# In[]:
def kMeans_onlineMode(clusteringData, num_cluster=5):
    patterns = np.array(clusteringData[['x','y']], dtype=np.float)
    #class_data = np.array(clusteringData['class'], dtype=np.int)
    P = np.size(patterns, 0) # pattern size
    idx = np.random.randint(P, size=num_cluster)
    W = patterns[idx,:]
    z_old = np.inf
    while True:
        b = np.zeros((P,num_cluster))
        for n in range(P):
            idx = np.argmin(np.linalg.norm(patterns[n]-W,axis=1))
            b[n,idx] = 1
        z=0
        for n in range(P):
            for i in range(num_cluster):
                if b[n,i]==1:
                    z = z + np.linalg.norm(patterns[n]-W[i])**2
            for i in range(num_cluster):
                if np.sum(b[:,i],axis=0) == 0:
                    print('Error: The random initial centers dont converge to given the number of clusters')
                    return np.inf,np.nan,np.nan
                else:
                    W[i,:] = np.sum(patterns[b[:,i]==1],axis=0)/np.sum(b[:,i],axis=0)


        if z_old<=z:
            break
        z_old=z
    return z,b,W

# In[]:
error,b,W = kMeans_onlineMode(data,5)
plotClusteringGraph(data, W, b, "Clustering in online mode with 5 centers", "part1b_N5")

# In[]:
error,b,W = kMeans_onlineMode(data,10)
plotClusteringGraph(data, W, b, "Clustering in online mode with 10 centers", "part1b_N10")

# In[]:
error,b,W = kMeans_onlineMode(data,15)
plotClusteringGraph(data, W, b, "Clustering in online mode with 15 centers", "part1b_N15")

# In[]:
def SOM(somData, num_neurons=5, alpha=0.7, sigma=1, beta1=0.7, beta2=0.7):
    patterns = np.array(somData[['x','y']], dtype=np.float)
    #class_data = np.array(somData['class'], dtype=np.int)
    t = 0
    P = np.size(patterns, 0) # pattern size
    idx = np.random.randint(P, size=num_neurons)
    W = patterns[idx,:]
    while True:
        np.random.shuffle(patterns)
        for n in range(P):
            idx = np.argmin(np.linalg.norm(patterns[n]-W,axis=1))
            for i in range(num_neurons):
```

```python
            neig_func = np.exp(-(np.linalg.norm(W[i,:]-W[idx,:])/sigma)**2)
            delta_w = alpha*neig_func*(patterns[n,:]-W[i,:])
            W[i,:] = W[i,:] + delta_w
        #print(W)
        if t==20:
            b = np.zeros((P,num_neurons))
            for n in range(P):
                idx = np.argmin(np.linalg.norm(patterns[n]-W,axis=1))
                b[n,idx] = 1
            break
        sigma = beta1*sigma
        alpha = beta2*alpha
        t = t + 1
    return patterns,b,W

# In[]:
shuffled_data,b,W = SOM(data,5, sigma=5/10)
plotClusteringGraph(shuffled_data, W, b, "Self organizing map with 5 neurons", "part1c_N5")

# In[]:
shuffled_data,b,W = SOM(data,10,sigma=10/10)
plotClusteringGraph(shuffled_data, W, b, "Self organizing map with 10 neurons", "part1c_N10")

# In[]:
shuffled_data,b,W = SOM(data,15,sigma=15/10)
plotClusteringGraph(shuffled_data, W, b, "Self organizing map with 15 neurons", "part1c_N15")

# In[]:
def cal_dist(data):
    data = data[:,[0,1]]
    N = np.size(data,0)
    total=0
    for n in range(N-1):
        total += np.linalg.norm(data[n+1,:]-data[n,:])
    total += np.linalg.norm(data[N-1,:]-data[0,:])
    return total

def plotSOM_TSPGraph(data, w_star, b_star, title, name="graph"):
    patterns = data[:,[0,1]]
    N = np.size(b_star,1)
    plt.figure()
    for n in range(N):
        plt.scatter(w_star[n,0],w_star[n,1], s=20, label=n+1,color="red")
    plt.plot(patterns[:,0], patterns[:,1],'bo-')
    plt.title(title)
    plt.savefig("{0}.png".format(name))
# ### Part 2:
# In[]:
def SOM_TSP(somData, num_neurons=81, neigFunc=0, alpha=1, sigma=1, beta1=0.8, beta2=0.8):
    patterns = np.array(somData[['x','y']], dtype=np.float)
    patterns = np.c_[patterns,np.linspace(1,81,81)]
    t = 0
    P = np.size(patterns, 0) # pattern size
    W = np.array([[np.random.uniform(0.0, patterns[:,0].max()), np.random.uniform(0.0, patterns[:,1].max()),i] for i in range(num_neurons)])
    while True:
        np.random.shuffle(patterns)
        for n in range(P):
            idx = np.argmin(np.linalg.norm(patterns[n,[0,1]]-W[:,[0,1]],axis=1))
            for i in range(num_neurons):
                if neigFunc==0:
                    neig_func = np.exp(-(np.linalg.norm(W[i,[0,1]]-W[idx,[0,1]])/sigma)**2)
                else:
                    d = np.min([np.abs(i-idx), num_neurons-np.abs(i-idx)])
                    neig_func = np.exp(-(d/sigma)**2)
        print(t)
        if t==100:
            b = np.zeros((P,num_neurons))
            ordering = np.zeros((P,1))
            for n in range(P):
                idx = np.argmin(np.linalg.norm(patterns[n,[0,1]]-W[:,[0,1]],axis=1))
                b[n,idx] = 1
                ordering[n]=idx
            patterns = np.c_[patterns,ordering]
            patterns = patterns[patterns[:,3].argsort()]
            patterns = np.vstack([patterns, patterns[0,:]])
            break
        sigma = beta1*sigma
        alpha = beta2*alpha
        t = t + 1
    return patterns,b,W

# In[]:
optimum_route = np.array([1,6,24,58,37,67,69,46,51,23,33,44,63,34,
    43,13,81,75,27,19,71,35,15,8,66,65,52,79,
    10,40,9,45,73,49,28,22,38,36,41,56,78,11,
    12,4,72,59,21,25,68,18,60,77,3,17,54,14,
    31,20,57,16,32,39,26,30,76,29,55,53,2,7,
    48,42,70,5,50,61,47,62,80,74,64])
optimum_order = np.array(data_tsp[['x','y']], dtype=np.float)
optimum_order = optimum_order[optimum_route-1,:]
optimum_order = np.vstack([optimum_order, optimum_order[0,:]])
plt.plot(optimum_order[:,0], optimum_order[:,1],'bo-')
plt.title("Optimum Route")
plt.savefig("{0}.png".format("part2_optimum"))
dist = cal_dist(optimum_order)

# In[]:
num_city = data_tsp.index[:].size
ordered_data,b,W = SOM_TSP(data_tsp,num_city*5, neigFunc=0, alpha=1, sigma=0.5, beta1=0.7, beta2=0.8)
plotSOM_TSPGraph(ordered_data, W, b, "Self organizing map with 81*5 neurons - GK", "part2a")
dist = cal_dist(ordered_data)

# In[]:
ordered_data,b,W = SOM_TSP(data_tsp,num_city*5, neigFunc=1, sigma=5*num_city/10)
```

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