MDM Ex2

October 11, 2020

1 Methods of Data Mining: Ex 2

1.1 Ahmed Bin Shafaat 795933

[4]:
```
#Importing libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import SpectralClustering
from sklearn.metrics import confusion_matrix
from sklearn.metrics import silhouette_score
from sklearn.metrics import normalized_mutual_info_score
from sklearn.metrics import davies_bouldin_score
from scipy.spatial.distance import euclidean, pdist, squareform
```

2 Exercise 1

[31]:
```
df_orig=pd.read_csv('spiral.txt',sep="\t", header=None) #reading text file
```

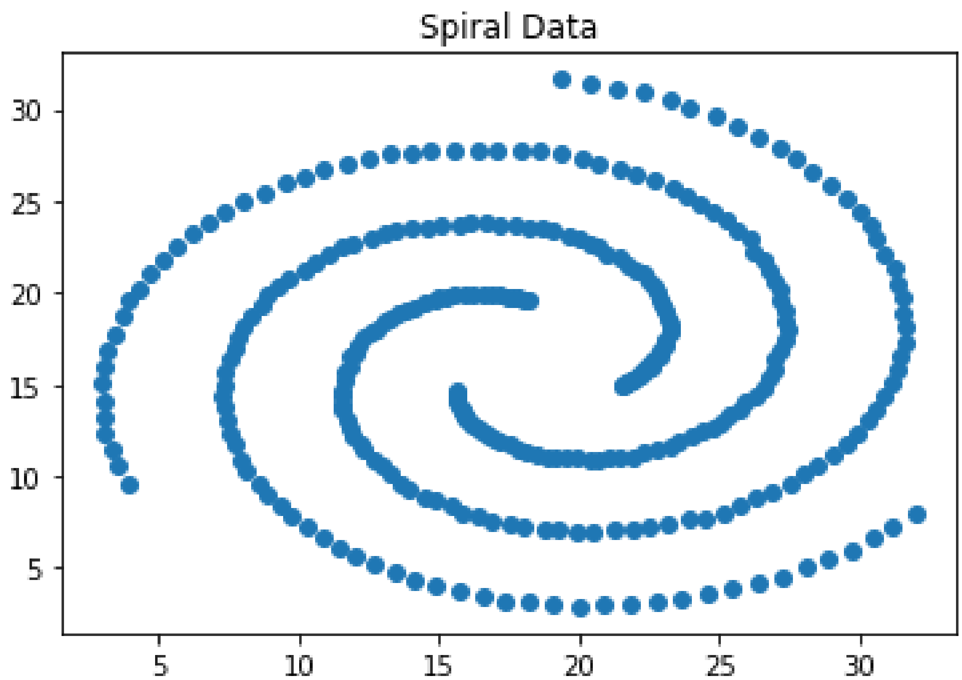
[32]:
```
orig_labels=df_orig.iloc[:,-1] # ground truth label
df_without_label=df_orig.iloc[:,:2] # Discarding ground truth label
```

[33]:
```
df_without_label.head(5) #showing first five rows after discarding the ground
 ↪truth label
```

[33]:

	0	1
0	31.95	7.95
1	31.15	7.30
2	30.45	6.65
3	29.70	6.00
4	28.90	5.55

```
[49]: plt.scatter(df_without_label.iloc[:,0:1],df_without_label.iloc[:,1:2]) #␣
      ↪Plotting the data
      plt.title("Spiral Data")
      plt.show()
```




```
[50]: df_without_label.describe() # Data has different mean and std so we will␣
      ↪standardize the data
```

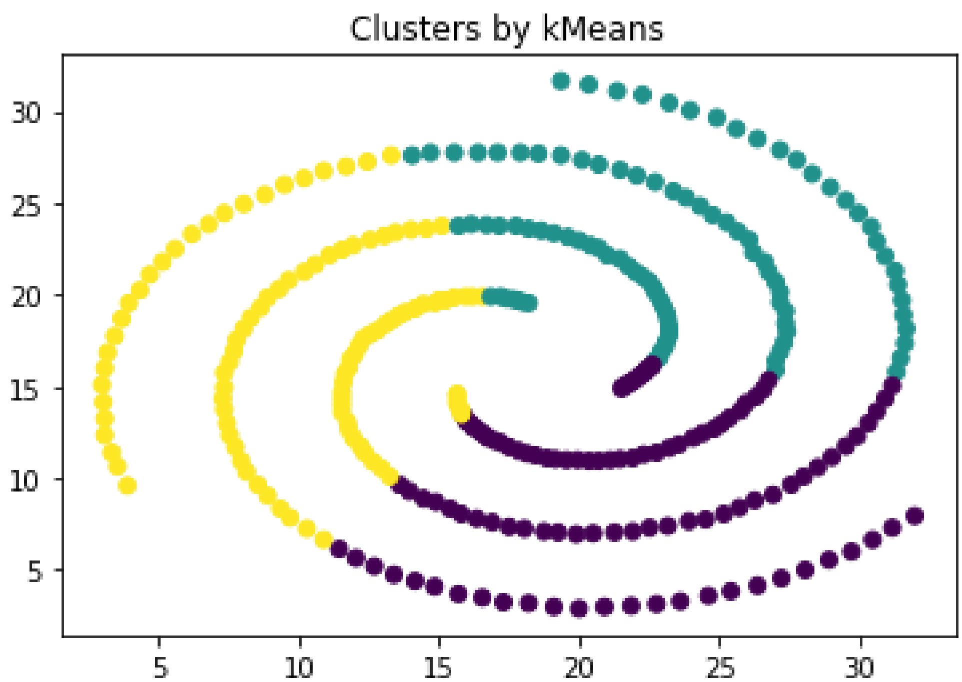
[50]:

	0	1
count	312.000000	312.000000
mean	18.408173	16.344712
std	7.299923	6.867232
min	3.000000	2.900000
25%	12.912500	11.337500
50%	18.325000	16.050000
75%	23.400000	21.362500
max	31.950000	31.650000

2.1 Performing KMeans Clustering

```
[51]: kmeans = KMeans(n_clusters=3, random_state=10)
      kmeans=kmeans.fit(df_without_label) # performing kmeans clustering
      KMclusters=kmeans.predict(df_without_label)
```

```
plt.scatter(df_without_label.iloc[:,0], df_without_label.iloc[:,1],␣
 ↪c=KMclusters)
plt.title("Clusters by kMeans")
plt.show()
```



2.1.1 Metric Scores

[52]:
```
print("Silhouette Scores: ",silhouette_score(data_scaled, np.
 ↪array(orig_labels)))
print("Normalized Mutual Info score:␣
 ↪",normalized_mutual_info_score(orig_labels, KMclusters))
print("Davies Bouldin Scores: ",davies_bouldin_score(data_scaled, np.
 ↪array(orig_labels)))
```

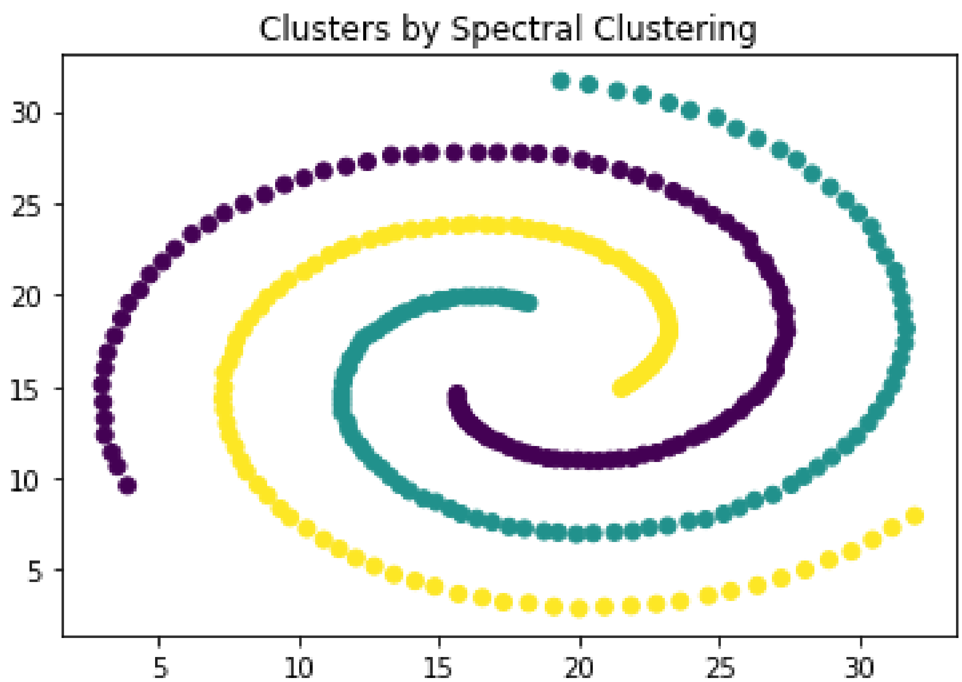
```
Silhouette Scores:  0.0013930072268410292
Normalized Mutual Info score:  0.0004200321266529102
Davies Bouldin Scores:  5.819660572385966
```

2.2 Spectral Clustering with default values of gamma and normalized laplacian

[53]:
```
SC = SpectralClustering(n_clusters=3,gamma=1.0 ,affinity='rbf')
```

[54]:
```
SCclusters=SC.fit_predict(df_without_label) # performing Spectral Clustering
```

```
plt.scatter(df_without_label.iloc[:,0], df_without_label.iloc[:,1],␣
    ↪c=SCclusters)
plt.title("Clusters by Spectral Clustering")
plt.show()
```



[55]:
```
print("Silhouette Scores: ",silhouette_score(data_scaled, np.
    ↪array(orig_labels)))
print("Normalized Mutual Info score:␣
    ↪",normalized_mutual_info_score(orig_labels, SCclusters))
print("Davies Bouldin Scores: ",davies_bouldin_score(data_scaled, np.
    ↪array(orig_labels)))
```

```
Silhouette Scores:  0.0013930072268410292
Normalized Mutual Info score:  1.0
Davies Bouldin Scores:  5.819660572385966
```

2.2.1 Finding optimal value of gamma

[56]:
```
gamma = range(3, 10, 1)
```

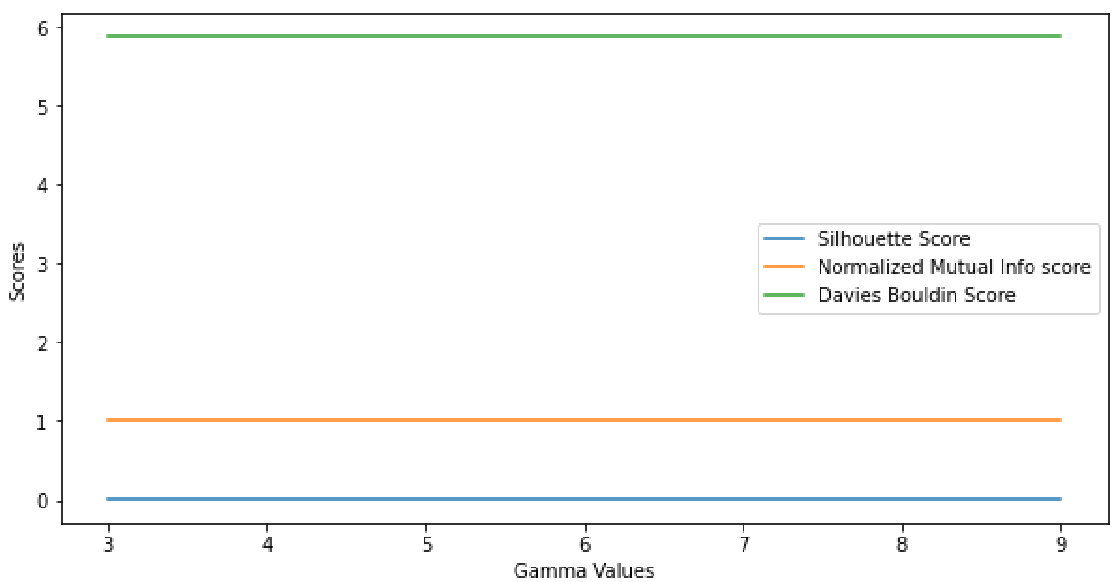
[67]:
```
SS = []
NMI=[]
DBS=[]
for g in gamma:
```

```
    SC = SpectralClustering(n_clusters=3,gamma=g ,affinity='rbf')
    SCclusters=SC.fit_predict(df_without_label)
    SS.append(silhouette_score(df_without_label, np.array(orig_labels)))
    NMI.append(normalized_mutual_info_score(orig_labels, SCclusters))
    DBS.append(davies_bouldin_score(df_without_label, np.array(orig_labels)))
```

[59]:
```
print("Silhouette Scores: ",SS)
print("Normalized Mutual Info score: ",NMI)
print("Davies Bouldin Scores: ",DBS)
```

```
Silhouette Scores:  [0.001344297344277985, 0.001344297344277985,
0.001344297344277985, 0.001344297344277985, 0.001344297344277985,
0.001344297344277985, 0.001344297344277985]
Normalized Mutual Info score:  [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Davies Bouldin Scores:  [5.882022552277642, 5.882022552277642,
5.882022552277642, 5.882022552277642, 5.882022552277642, 5.882022552277642,
5.882022552277642]
```

[62]:
```
plt.figure(figsize=(10,5))
plt.plot(gamma, SS ,label = "Silhouette Score")
plt.plot(gamma, NMI ,label = "Normalized Mutual Info score")
plt.plot(gamma, DBS ,label = "Davies Bouldin Score")
plt.ylabel('Scores')
plt.xlabel('Gamma Values')
plt.legend()
plt.show()
```



2.2.2 Discussion:

I have performed KMeans and Spectral clustering. Before clustering, I discarded the ground truth label from the data. For spectral clustering, I used RBF Gaussian kernel and default values of gamma. I used SKlearn library which is using normalized laplacian matrix. KMeans yielded good Davies Bouldin Score but as far as peroformance measure is concerned in other metrics' perspective, Silhouette index indicated that clusters produced by KMeans are overlapping as we are getting a Silhouette score around zero which indicates overlapping clusters. Normalized mutual info score is again, very poor for kmeans which is around zero too.

For Spectral clustering, I got almost same results for Davies Bouldin and Silhouette indexed which shows no improvement but NMI score showed a greate increase which is an indication of better labelling. I tried to find optimal value of gamma in spectral clustering's case but different values of gamma resulted in similar scores for Davies Bouldin, Normalized Mutual Info and Silhouette which are shown above. Due to change in NMI score, I would rank Spectral clustering better than KMeans. And according to my resulting scores, NMI captures performance of the algorithm more accurately as other two are not changing according to the change in the values of hyperparameters.

3 Excercise 2

[100]:
```
#Computing Kernel Matrix
def compute_kernel_matrix(data,sigma):
    length=len(data)
    kernel_matrix=np.zeros((length,length))
    for idx, i in enumerate(data):
        for index, j in enumerate(data):
            if(idx==index):
                index=index+1
                continue;
            num = -1 * ( np.abs( (data[idx][0] - data[index][0]) +␣
 ↪(data[idx][1] - data[index][1]) )**2 )
            denom = 2 * sigma**2
            kernel_matrix[idx][index] = np.exp(np.divide( num ,denom))
    return kernel_matrix
```

[101]:
```
#Computing Clustering Matrix
def compute_Clustering_matrix(data,c):
    length=len(data)
    clustering_matrix=np.zeros((length,length))
    clusters=[]
    if c=='k':
        clusters=KMclusters
    elif c=='s':
        clusters=SCclusters
    for idx, i in enumerate(data):
        for index, j in enumerate(data):
            if ( clusters[idx] == clusters[index]):
                clustering_matrix[idx][index]=1
```

```
            else:
                clustering_matrix[idx][index]=0
    return clustering_matrix
```

[115]:
```
#Computing t
def compute_taa_metric(sigma,c='k'):
    taa=[]
    scaler = StandardScaler()
    data=scaler.fit_transform(df_without_label)
    for s in sigma:
        kernel=compute_kernel_matrix(data,s)
        clustering=compute_Clustering_matrix(data,c=c)
        upper_sum = np.triu(kernel).sum()-np.trace(kernel)
        lower_sum = np.tril(kernel).sum()-np.trace(kernel)
        kernel_sum=upper_sum + lower_sum
        num_sum=0
        for idx, i in enumerate(kernel):
            for index, j in enumerate(kernel):
                if(idx!=index):
                    num_sum+= clustering[idx][index] * kernel[idx][index]
        taa.append(np.sum(num_sum / kernel_sum) / 312)
    return taa
```

3.1 a):

3.2 Considering KMeans Clustering

[116]:
```
sigma=[1,2,3,4,5,6,7]
taa=compute_taa_metric(sigma,'k')
```

[117]:
```
taa
```

[117]:
```
[0.0015009657425071525,
 0.0012843435650395636,
 0.0011832146745020795,
 0.0011364224016613103,
 0.0011121582917692542,
 0.001098210766001285,
 0.0010895262041301383]
```

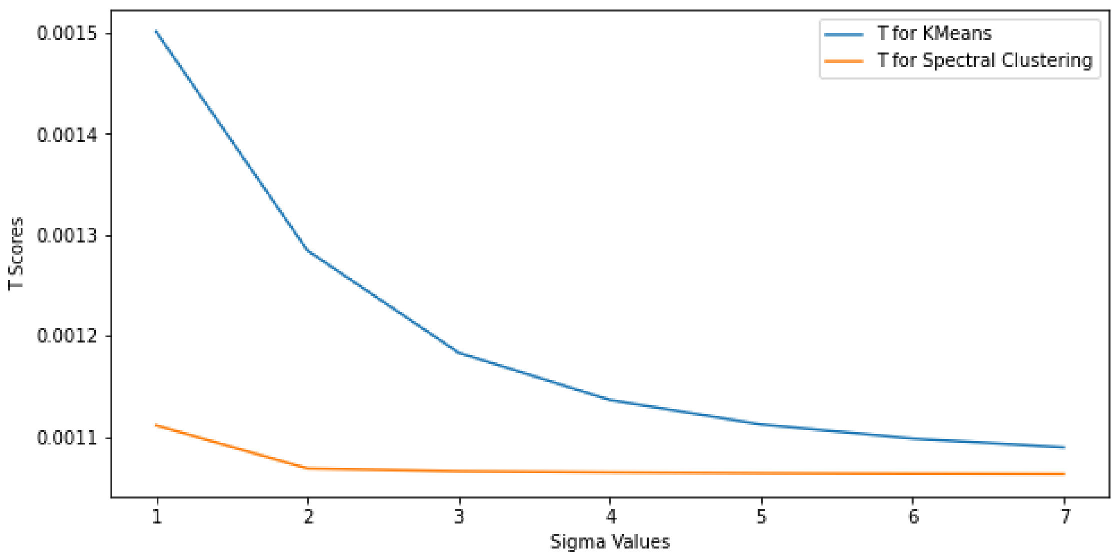
3.3 Considering Spectral Clustering for gamma = 1.0

[118]:
```
taa_=compute_taa_metric(sigma,'s')
```

[119]:
```
taa_
```

```
[119]: [0.0011112758431341917,
 0.0010686035912216836,
 0.0010658273026050002,
 0.0010647279061649167,
 0.0010639880373056044,
 0.0010634846089871979,
 0.0010631376924637912]
```

```
[120]: plt.figure(figsize=(10,5))
plt.plot(sigma, taa ,label = "T for KMeans")
plt.plot(sigma, taa_ ,label = "T for Spectral Clustering")
plt.ylabel('T Scores')
plt.xlabel('Sigma Values')
plt.legend()
plt.show()
```



One thing which is obvius from plotting the τ and Silhouette and Davies-Bouldin index is, for varying values of gamma,later two almost remained constant while τ is decreasing with increasing values of sigma. So what sigma is doing here, for larger values of the sigma, we get larger kernel matrix to capture enough of the function's energy which resultantly is making the kernel more "fat" to separate the data linearly in higher dimnensions. As far as comparison between newly computed τ, Silhouette and Davies-Bouldin index is concerned, I think τ is more informative as we can infer that for larger sigma values, kernel function is able to transform the data in such a way that more distinct clusters could be formed, which resultantly decrease the τ score, which again is a good thing. Apparently, by the plot, τ is more informative than the previous two metrics as one can atleast infer some results from the plot.

3.4 b):

One clear disadvantage which is very obvius in using τ is this algorithm is pretty complex in its nature. One more thing is we are dependant on the optimal value of sigma to obtain better cluster leballing. Since result is dependant on a variable, results are not always constant in nature and they are actually supposed to be fine tuned.

3.5 c):

I will use pairwise distance between data samples to compute an adjacency matrix and will use this to compute τ.

[121]:
```
def similarity_func(u, v):
    return 1/(1+euclidean(u,v))
```

[122]:
```
dists = pdist(data_scaled, similarity_func)
kernel = squareform(dists)
taa=0
clustering=compute_Clustering_matrix(data_scaled,c='k')
upper_sum = np.triu(kernel).sum()-np.trace(kernel)
lower_sum = np.tril(kernel).sum()-np.trace(kernel)
kernel_sum=upper_sum + lower_sum
num_sum=0
for idx, i in enumerate(kernel):
    for index, j in enumerate(kernel):
        if(idx!=index):
            num_sum+= clustering[idx][index] * kernel[idx][index]
taa=np.sum(num_sum / kernel_sum) / 312
```

[123]:
```
taa
```

[123]:
```
0.0013547833506971684
```

4 Excercise 3

4.1 a):

[124]:
```
def lift(n,frXC,frX,frC):
    return (n*frXC)/(frX*frC)
```

[125]:
```
def leverage(n,frXC,frX,frC):
    return frXC/n - ((frX/n)*(frC/n))
```

[130]:
```
rules = [[1,'smoking → heart disease',300,125],
        [2,'stress → heart disease',500,150],
        [3,'sports → ¬ heart disease',500,400],
        [4,'coffee → ¬ heart disease',342,240],
        [5,'natural product → ¬ heart disease ',2,2],
```

```
        [6,'female → ¬ heart disease',500,352],
        [7,'female, stress → heart disease',260,100],
        [8,'chocolate, bananas → heart disease',120,32],
        [9,'smoking, coffee → heart disease',240,100],
        [10,'smoking, sports → heart disease',80,32],
        [11,'stress, smoking → heart disease',200,100],
        [12,'female, sports → ¬ heart disease',251,203]]
```

[127]:
```
df_rules = pd.DataFrame(rules, columns = ['num','rule','frX','frXC'])
```

[128]:
```
number_of_ppl=1000
frC=300 # because 30% of population has heart desease
for line in range(df_rules.shape[0]):
    df_rules.loc[line,'lift'] = lift(number_of_ppl,df_rules.
 ↪loc[line,'frXC'],df_rules.loc[line,'frX'],frC)
    df_rules.loc[line,'leverage'] = leverage(number_of_ppl,df_rules.
 ↪loc[line,'frXC'],df_rules.loc[line,'frX'],frC)
```

[129]:
```
df_rules
```

[129]:

	num	rule	frX	frXC	lift	leverage
0	1	smoking → heart disease	300	125	1.388889	0.0350
1	2	stress → heart disease	500	150	1.000000	0.0000
2	3	sports → ¬ heart disease	500	400	2.666667	0.2500
3	4	coffee → ¬ heart disease	342	240	2.339181	0.1374
4	5	natural product → ¬ heart disease	2	2	3.333333	0.0014
5	6	female → ¬ heart disease	500	352	2.346667	0.2020
6	7	female, stress → heart disease	260	100	1.282051	0.0220
7	8	chocolate, bananas → heart disease	120	32	0.888889	-0.0040
8	9	smoking, coffee → heart disease	240	100	1.388889	0.0280
9	10	smoking, sports → heart disease	80	32	1.333333	0.0080
10	11	stress, smoking → heart disease	200	100	1.666667	0.0400
11	12	female, sports → ¬ heart disease	251	203	2.695883	0.1277

Since rule no. 8, on 7th line, since numbering is starting from 0, has negative leverage, so we will prune it out.

4.2 b):

4.3 I do not know

4.4 c): I do not know

4.5 d): I do not know

5 Excercise 4

5.1 a): "I don't know"

5.2 b): "I don't know"

6 Excercise 5:

6.1 "I don't know"