Sheet4_Evaluation

March 24, 2019

```
In [0]: from sklearn.cluster import KMeans
        import numpy as np
        kvalues=[2,3,4,5,6,2,2,3,3,4,4,5,5,6,6]
        printList =["Kmeans clustering when k =2","Kmeans clustering when k =3","Kmeans cluste
                   "Kmeans clustering when k =2 and gamma=0.01","Kmeans clustering when k =2 an
                    ,"Kmeans clustering when k =4 and gamma=0.01","Kmeans clustering when k =4
                    ,"Kmeans clustering when k =6 and gamma =0.01","Kmeans clustering when k =
```

1 a. Using Kmeans: set K=2,3,4,5,6. Report different clustering results.

```
In [0]: #Question 1
        data_array = [[2,4],[4,4],[3,4],[5,4],[5,6],[5,8],[6,4],[6,5],[6,7],[7,3],[7,4],[8,2],
        data_array = np.array(data_array);
        kmeanLabels = []
        for i in range(2,7):
         kmeans = KMeans(n_clusters=i, random_state=0).fit(data_array)
         kmeanLabels.append(kmeans.labels_)
        #kmeanLabels = np.array(kmeanLabels)
        print(np.array(kmeanLabels))
```

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 0 0 0 0 0]
 [3 3 3 3 3 3 1 3 3 1 1 1 1 0 0 0 0 0 0 2 0 2 2 2]
 [4 4 4 4 3 3 1 1 3 1 1 1 1 0 0 0 0 0 0 2 2 2 2 2]
 [4 4 4 4 3 3 1 3 3 1 1 1 1 2 2 2 2 2 0 0 0 0 5 5]]
```

2 b)i)Use RBF kernel with gamma = {0.01,0.1}. Report the Report

3 different clustering results.

```
In [0]: from sklearn.cluster import SpectralClustering
        for i in range(2,7):
         #we calculate RBF -->np.exp(-gamma * d(X,X) ** 2)
         clustering1 = SpectralClustering(gamma=0.01,n_clusters=i,assign_labels="discretize",ra
         clustering2 = SpectralClustering(gamma=0.1,n_clusters=i,assign_labels="discretize",ra
```

```
         kmeanLabels.append(clustering1.labels_)
         kmeanLabels.append(clustering2.labels_)
        print(np.array(kmeanLabels))
```

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 0 0 0 0 0]
 [3 3 3 3 3 3 1 3 3 1 1 1 1 0 0 0 0 0 0 2 0 2 2 2]
 [4 4 4 4 3 3 1 1 3 1 1 1 1 0 0 0 0 0 0 2 2 2 2 2]
 [4 4 4 4 3 3 1 3 3 1 1 1 1 2 2 2 2 2 0 0 0 0 5 5]
 [1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 0 1 1 0 1 1 2 2 0 0 0 2 0 0 2 0 2 2 2]
 [1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 2 2 2 2 2]
 [3 3 3 3 3 0 1 3 0 1 1 1 1 0 0 0 1 0 0 2 2 2 2 2]
 [1 1 1 1 1 1 3 3 1 3 3 3 3 0 0 0 0 0 0 2 2 2 2 2]
 [4 4 4 4 1 1 3 1 1 3 3 3 3 0 0 0 3 0 0 2 2 2 2 2]
 [1 1 1 1 4 4 3 4 4 3 3 3 3 0 0 0 0 0 0 2 2 2 2 2]
 [2 2 2 2 3 3 2 3 3 2 2 2 1 5 0 0 1 0 0 4 4 4 4 4]
 [1 1 1 1 4 4 3 4 4 3 3 3 5 5 0 0 5 0 0 2 0 2 2 2]]
```

4 ii. Use Similarity graph as the {3,5}-NN graph. Where Sim(xi,xj)=1

5 iff xj is one of the nearest three points to xi (or vise versa). Report

6 different clustering results.

```
In [0]: from sklearn.cluster import SpectralClustering
        for i in range(2,7):
         clustering1 = SpectralClustering(affinity='nearest_neighbors',n_neighbors=3,n_cluster
         kmeanLabels.append(clustering1.labels_)
         clustering2 = SpectralClustering(affinity='nearest_neighbors',n_neighbors=5,n_cluster
         kmeanLabels.append(clustering2.labels_)
        print(np.array(kmeanLabels))
        kmeanLabels=np.array(kmeanLabels)
```

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 0 0 0 0 0]
 [3 3 3 3 3 3 1 3 3 1 1 1 1 0 0 0 0 0 0 2 0 2 2 2]
 [4 4 4 4 3 3 1 1 3 1 1 1 1 0 0 0 0 0 0 2 2 2 2 2]
 [4 4 4 4 3 3 1 3 3 1 1 1 1 2 2 2 2 2 0 0 0 0 5 5]
 [1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 0 1 1 0 1 1 2 2 0 0 0 2 0 0 2 0 2 2 2]
 [1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 2 2 2 2 2]
 [3 3 3 3 3 0 1 3 0 1 1 1 1 0 0 0 1 0 0 2 2 2 2 2]
 [1 1 1 1 1 1 3 3 1 3 3 3 3 0 0 0 0 0 0 2 2 2 2 2]
```

```
[4 4 4 4 1 1 3 1 1 3 3 3 3 0 0 0 3 0 0 2 2 2 2 2]
[1 1 1 1 4 4 3 4 4 3 3 3 3 0 0 0 0 0 0 2 2 2 2 2]
[2 2 2 2 3 3 2 3 3 2 2 2 1 5 0 0 1 0 0 4 4 4 4 4]
[1 1 1 1 4 4 3 4 4 3 3 3 5 5 0 0 5 0 0 2 0 2 2 2]
[1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0]
[1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0]
[2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 1 1 1 1 1 1]
[2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 1 1 1 1 1]
[3 3 3 3 2 2 2 2 2 2 2 2 0 0 0 0 0 0 1 1 1 1 1 1]
[1 1 1 1 1 1 3 1 1 3 3 3 3 0 0 0 0 0 0 2 2 2 2 2]
[1 1 1 1 2 2 4 2 2 4 4 4 4 0 0 0 0 0 3 3 3 3 3 3]
[1 1 1 1 2 2 4 2 2 4 4 4 4 0 0 0 0 0 0 3 3 3 3 3]
[1 1 1 1 2 2 5 2 2 5 5 5 5 0 0 0 0 0 4 4 4 3 3 3]
[2 2 2 2 4 4 1 4 4 1 1 1 5 5 0 0 5 0 0 3 3 3 3 3]]
```

7 Needed Functions

8 C)Compute the external measures we studied such as

8.1 1. Conditional Entropy

In [0]:
```
trueCluster=[0 ,0 ,0 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,2 ,2 ,2 ,2 ,2 ,2 ,2 ,2 ,2 ,2 ,2 ]
startof0 =0;
endof0 = 3;
startof1 =3;
endof1 = 13;
startof2 = 13;
endof2 = 24;
def Euclidean(a,b):
     return math.sqrt((a[0]-b[0])**2 +(a[1]-b[1])**2)
def mapClusterwithtrueOne(clusters,k):
  mp =np.zeros((3,k))
  for i in range(0,3):
    for j in range(0,k):
      mp[i][j]=0;
  for i in range(0,24):
    mp[trueCluster[i]][clusters[i]]=mp[trueCluster[i]][clusters[i]]+1;
  return mp
def totalSum(arr,k):
  sum=0;
  for i in range(0,k):
    sum+=arr[i];
  return sum
```

In [0]:
```
import math
def compute_Entropy(clusters,k):
    mymap = mapClusterwithtrueOne(clusters,k)
```

```
    for i in range(0,3):
      print("Entropy for cluster  ",i+1)
      sum = totalSum(mymap[i],k)
      entropy =0;
      for j in range(0,k):
        if(mymap[i][j] ==0):
           entropy -=(mymap[i][j]/sum)
        else:
           entropy -=(mymap[i][j]/sum)*math.log(mymap[i][j]/sum,2)
      entropy = entropy * (sum/24)
      print("=",entropy)
```

9 2. Purity

```
In [0]: def compute_Purity(clusters,k):
          mymap = mapClusterwithtrueOne(clusters,k)
          totalpurity=0
          for i in range(0,3):
            purity=0
            sum = totalSum(mymap[i],k)
            purity = np.amax(mymap[i])/sum
            totalpurity += (sum/24)*purity
          print("Purity =",totalpurity)
```

10 3. Pairwise measures (Jaccard and Rand index)

```
In [0]: def Pairwise(clusters,k):
        TP=0;
        FP = 0
        FN = 0
        TN = 0
        for i  in range(0,24):
          for j in range(i+1,24):
            if clusters[i] == clusters[j]:
              if trueCluster[i] ==  trueCluster[j]:
                TP +=1
              else:
                FP +=1
            else:
              if trueCluster[i] ==  trueCluster[j]:
                FN +=1
              else:
                TN +=1
        rand = (TP + TN)/(TP + TN +FP + FN)
        jaccard = (TP)/(FN + TP + FP)
        print("Rand = ",rand)
        print("Jaccard = ",jaccard)
```

11 4. MaxMatching

```
In [0]: def MaxMatching(clusters):
          mymap = mapClusterwithtrueOne(clusters,3)
          mx = 0
          mx = max(mymap[0][0]+mymap[1,1]+mymap[2,2],mymap[0][0]+mymap[1,2]+mymap[2,1],
                          mymap[0][1]+mymap[1,0]+mymap[2,2],mymap[0][1]+mymap[1,2]+mymap[2,0],
                          mymap[0][2]+mymap[1,1]+mymap[2,0],mymap[0][2]+mymap[1,0]+mymap[2,1])
          print("MaxMatching = ",mx/24)
```

12 5. FMeasure

```
In [0]: def compute_FMeasure(clusters,k):
          mymap = mapClusterwithtrueOne(clusters,k)
          F = 0
          for i in range(0,3):
            sum = totalSum(mymap[i],k)
            prec =np.amax(mymap[i])/sum
            indx = np.argmax(mymap[i])
            recall = np.amax(mymap[i])/totalSum(mymap[:,indx],3)
            F += (2*prec*recall)/(prec+recall)
          print("FMeasure= ",F/k)
```

13 Internal Measure

14 BetaCV

```
In [0]: #compact ---> between cluster and itself
        def win(labels):
            w = 0
            for i in range(0,24):
               for j in range(0,24):
                 if labels[i] == labels[j] :
                   w += Euclidean(data_array[i],data_array[j])
            return w/2
         #between cluster and others
        def wout(labels):
            w = 0
            for i in range(0,24):
               for j in range(0,24):
                 if labels[i] != labels[j] :
                   w += Euclidean(data_array[i],data_array[j])
```

```
        return w/2
    def betaCV(labels,k):
        ni = np.zeros((k,1))
    # cluster sizes
        for i in range(0,24):
           ni[labels[i]] += 1
        Nin = 0
        for i in range(0,k):
           Nin += ni[i]*(ni[i]-1)
        Nin *= 0.5
        Nout = 0
        for i in range(0,k):
         for j in range(i+1,k):
           Nout += ni[i] * ni[j]
        print(Nout*win(labels) / (Nin*wout(labels)))
```

In [0]:

15 Normalized cut

In [0]:
```
def Normalized_Cut(labels,k):
    nc = 0
    for i in range(0,k):
      w1 = 0
      w2 = 0
      for j in range(0,24):
        if labels[j] != i :
           continue
      for k in range(0,24):
         dis = Euclidean(data_array[j],data_array[k])
         if labels[k] == labels[j] :
               w1 += dis
         else:
             w2 += dis
    nc += 1/((w1/w2)+1)
    print("Normalized cut",nc)
```

16 Calculation

In [0]:
```
for i in range(0,len(kmeanLabels)):
    print(printList[i])
    print(kmeanLabels[i])
    compute_Entropy(kmeanLabels[i],kvalues[i])
    compute_Purity(kmeanLabels[i],kvalues[i])
    Pairwise(kmeanLabels[i],kvalues[i])
    compute_FMeasure(kmeanLabels[i],kvalues[i])
    betaCV(kmeanLabels[i],kvalues[i])
```

```
        Normalized_Cut(kmeanLabels[i],kvalues[i])
        if kvalues[i]==3:
            MaxMatching(kmeanLabels[i])
```

```
Kmeans clustering when k =2
[0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1]
Entropy for cluster   1
= 0.0
Entropy for cluster   2
= 0.0
Entropy for cluster   3
= 0.0
Purity = 1.0
Rand =  0.8913043478260869
Jaccard =  0.7744360902255639
FMeasure=  1.1222826086956523
[0.4324087]
Normalized cut 0.7498748080414097
Kmeans clustering when k =3
[1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 0 0 0 0 0]
Entropy for cluster   1
= 0.0
Entropy for cluster   2
= 0.19541483066220053
Entropy for cluster   3
= 0.4555971802602717
Purity = 0.75
Rand =  0.7391304347826086
Jaccard =  0.47058823529411764
FMeasure=  0.6282828282828282
[0.42794904]
Normalized cut 0.9414521502010061
MaxMatching =  0.625
Kmeans clustering when k =4
[3 3 3 3 3 3 1 3 3 1 1 1 1 0 0 0 0 0 0 2 0 2 2 2]
Entropy for cluster   1
= 0.0
Entropy for cluster   2
= 0.4166666666666667
Entropy for cluster   3
= 0.43342763960862674
Purity = 0.625
Rand =  0.7536231884057971
Jaccard =  0.423728813559322
FMeasure=  0.4974747474747474
[0.37818537]
```

```
Normalized cut 0.9632821669633898
Kmeans clustering when k =5
[4 4 4 4 3 3 1 1 3 1 1 1 1 0 0 0 0 0 0 2 2 2 2 2]
Entropy for cluster   1
= 0.0
Entropy for cluster   2
= 0.5397757684326342
Entropy for cluster   3
= 0.4555971802602717
Purity = 0.625
Rand =  0.782608695652174
Jaccard =  0.4339622641509434
FMeasure=  0.4626050420168067
[0.32170029]
Normalized cut 0.9414521502010061
Kmeans clustering when k =6
[4 4 4 4 3 3 1 3 3 1 1 1 1 2 2 2 2 2 0 0 0 0 5 5]
Entropy for cluster   1
= 0.0
Entropy for cluster   2
= 0.5670683531015339
Entropy for cluster   3
= 0.6851711387738941
Purity = 0.5416666666666667
Rand =  0.7463768115942029
Jaccard =  0.33962264150943394
FMeasure=  0.35813492063492064
[0.30170215]
Normalized cut 0.9939454427091529
Kmeans clustering when k =2 and gamma=0.01
[1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0]
Entropy for cluster   1
= 0.0
Entropy for cluster   2
= 0.19541483066220053
Entropy for cluster   3
= 0.0
Purity = 0.9583333333333333
Rand =  0.8297101449275363
Jaccard =  0.6666666666666666
FMeasure=  1.0873517786561266
[0.43636769]
Normalized cut 0.7135474642963268
Kmeans clustering when k =2 and gamma=0.1
[1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0]
Entropy for cluster   1
= 0.0
Entropy for cluster   2
```

```
= 0.0
Entropy for cluster   3
= 0.0
Purity = 1.0
Rand =  0.8913043478260869
Jaccard =  0.7744360902255639
FMeasure=  1.1222826086956523
[0.4324087]
Normalized cut 0.7498748080414097
Kmeans clustering when k =3 and gamma=0.01
[1 1 1 1 1 0 1 1 0 1 1 2 2 0 0 0 2 0 0 2 0 2 2 2]
Entropy for cluster   1
= 0.0
Entropy for cluster   2
= 0.571229414356112
Entropy for cluster   3
= 0.4555971802602717
Purity = 0.625
Rand =  0.644927536231884
Jaccard =  0.3146853146853147
FMeasure=  0.587719298245614
[0.51504319]
Normalized cut 0.8579134015834787
MaxMatching =  0.5
Kmeans clustering when k =3 and gamma=0.1
[1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 2 2 2 2 2]
Entropy for cluster   1
= 0.0
Entropy for cluster   2
= 0.19541483066220053
Entropy for cluster   3
= 0.4555971802602717
Purity = 0.75
Rand =  0.7391304347826086
Jaccard =  0.47058823529411764
FMeasure=  0.6282828282828282
[0.42794904]
Normalized cut 0.9414521502010061
MaxMatching =  0.625
Kmeans clustering when k =4 and gamma=0.01
[3 3 3 3 3 0 1 3 0 1 1 1 1 0 0 0 1 0 0 2 2 2 2 2]
Entropy for cluster   1
= 0.0
Entropy for cluster   2
= 0.6189480405113894
Entropy for cluster   3
= 0.6181027856723602
Purity = 0.5416666666666667
```

```
Rand =  0.6739130434782609
Jaccard =  0.29133858267716534
FMeasure=  0.4618055555555555
[0.44531629]
Normalized cut 0.9414521502010061
Kmeans clustering when k =4 and gamma=0.1
[1 1 1 1 1 1 3 3 1 3 3 3 3 0 0 0 0 0 0 2 2 2 2 2]
Entropy for cluster   1
= 0.0
Entropy for cluster   2
= 0.4045627476894453
Entropy for cluster   3
= 0.4555971802602717
Purity = 0.625
Rand =  0.7608695652173914
Jaccard =  0.4260869565217391
FMeasure=  0.5139705882352941
[0.37255019]
Normalized cut 0.9414521502010061
Kmeans clustering when k =5 and gamma =0.01
[4 4 4 4 1 1 3 1 1 3 3 3 3 0 0 0 3 0 0 2 2 2 2 2]
Entropy for cluster   1
= 0.0
Entropy for cluster   2
= 0.5670683531015339
Entropy for cluster   3
= 0.6181027856723603
Purity = 0.5416666666666667
Rand =  0.7391304347826086
Jaccard =  0.35135135135135137
FMeasure=  0.42142857142857143
[0.35104427]
Normalized cut 0.9414521502010061
Kmeans clustering when k =5 and gamma =0.1
[1 1 1 1 4 4 3 4 4 3 3 3 3 0 0 0 0 0 0 2 2 2 2 2]
Entropy for cluster   1
= 0.0
Entropy for cluster   2
= 0.5670683531015338
Entropy for cluster   3
= 0.4555971802602717
Purity = 0.5833333333333334
Rand =  0.7753623188405797
Jaccard =  0.41509433962264153
FMeasure=  0.4459383753501401
[0.31972652]
Normalized cut 0.9414521502010061
Kmeans clustering when k =6 and gamma =0.01
```

```
[2 2 2 2 3 3 2 3 3 2 2 2 1 5 0 0 1 0 0 4 4 4 4 4]
Entropy for cluster   1
= 0.0
Entropy for cluster   2
= 0.5670683531015338
Entropy for cluster   3
= 0.7685044721072276
Purity = 0.5416666666666667
Rand =  0.6956521739130435
Jaccard =  0.29411764705882354
FMeasure=  0.2876683501683502
[0.4019425]
Normalized cut 0.9414521502010061
Kmeans clustering when k =6 and gamma =0.1
[1 1 1 1 4 4 3 4 4 3 3 3 5 5 0 0 5 0 0 2 0 2 2 2]
Entropy for cluster   1
= 0.0
Entropy for cluster   2
= 0.717470039536401
Entropy for cluster   3
= 0.6851711387738941
Purity = 0.5
Rand =  0.7246376811594203
Jaccard =  0.2962962962962963
FMeasure=  0.3422619047619048
[0.31116792]
Normalized cut 0.9632821669633898
```

```
---------------------------------------------------------------------------

IndexError                                Traceback (most recent call last)

<ipython-input-99-3b05dc7557eb> in <module>()
      1
      2 for i in range(0,len(kmeanLabels)):
----> 3   print(printList[i])
      4   print(kmeanLabels[i])
      5   compute_Entropy(kmeanLabels[i],kvalues[i])


IndexError: list index out of range
```

In [0]: