Artificial Intelligence: Knowledge Representation and Planning

CM0472: Assignment 3 – Clustering

Team:

Gianluca Bison - [**870028**]

Ivanoiu Alexandru Paul - [**870383**]

Index:

Assignment:

Perform classification of the Semeion Handwritten Digit data set using:

1. **Latent Class Analysis** - [LCA]
2. **Mean Shift**
3. **Normalized Cut**

Mean Shift and **Normalized Cut** assume that the images are vectors in a *256* dimensional Euclidean space.

Provide the code and the extracted clusters as the number of clusters **k** varies from *5* to *15*.

For **LCA** and **Normalized Cut**, while for **Mean Shift** vary the kernel width.

For each value of k (or kernel width) provide the value of the Rand index:

$$R=2(a+b)/(n(n-1))$$

where:

a) n is the number of images in the dataset.

b) a is the number of pairs of images that represent the same digit and that are clustered together.

c) b is the number of pairs of images that represent different digits and that are placed in different clusters.

Explain the differences between the three models.

Tip: Bernoulli models can be visualized as a grayscale image to inspect the learned model.

Goal of this assignment:

The goal of this assignment is to use the **Semeion Handwritten Digit data set** and perform a classification using the 3 algorithms requested: LCA / Mean Shift / Normalized Cut.

For each classification algorithm we have to modify the number of k (**clusters**) and the **width band** in case of the mean shift and see the differences. For each k we also have to run the **Rand Index** and see the results and compare each classifier and see the advantages and disadvantages.

Also we have to say which of the k returns the best result overall.

In the end we have to show the performance and explain the ups and downs for each choice.

What does Clustering mean and what is the category that he belongs to?

Clustering it is basically an **unsupervised learning method**.

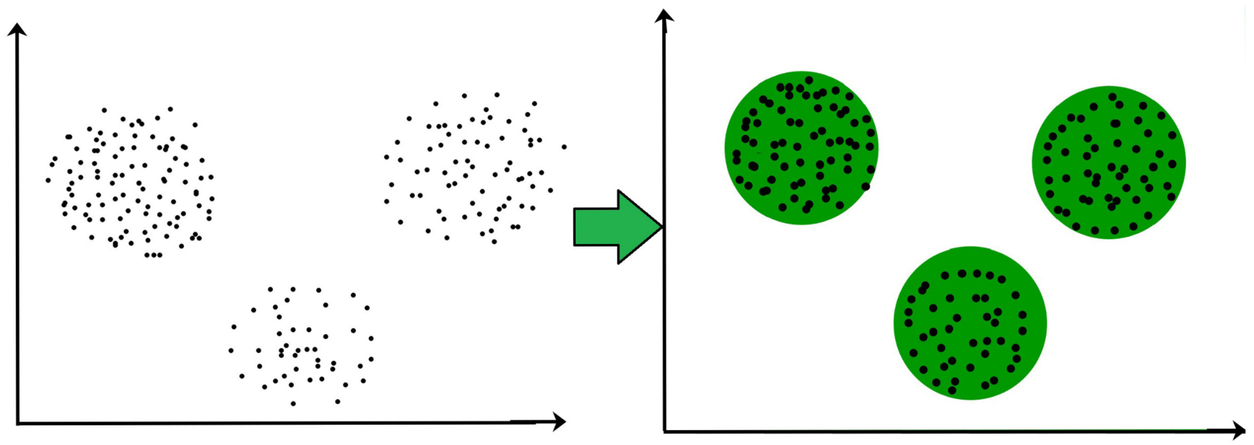
Unsupervised learning means the training of a machine using information that is not labeled or classified and pushes the algorithm to act on that information **without** a **teacher** as it happens on the **supervised learning method**.

So when you have the task to classify(group) data like in our assignment the algorithm has to check the similarities/distances between the data in the dataset. This time no training will be given to the machine as we know it happens in the supervised case, so it is used as a process to find structures that have a meaning in a set of examples.

As for the **clustering** part, the task of a **clustering algorithm** is to **divide** the population or the data point into a number of groups(**K**) such that the data points in the same group have the same characteristics, or better the data points are similar, the bigger the similarity inside the cluster the better, as for the points that are in different clusters they have to be as different as possible.

So basically it is a collection of objects that form groups based on their similarities and dissimilarities.

Below we can see an example of what does a clustering problem mean:



As we can see in the picture above we have a space and we have a data set of points that form 3 clusters in the end because as we can see they are 3 conglomerates that are very compact and so we can say that the **internal criterion** (all “objects” inside a cluster should be highly similar to each other) hold and the **external criterion** holds too (all “objects” outside a cluster should be highly dissimilar to the ones inside).

Bellow we can find the classical clustering problem:

Given :

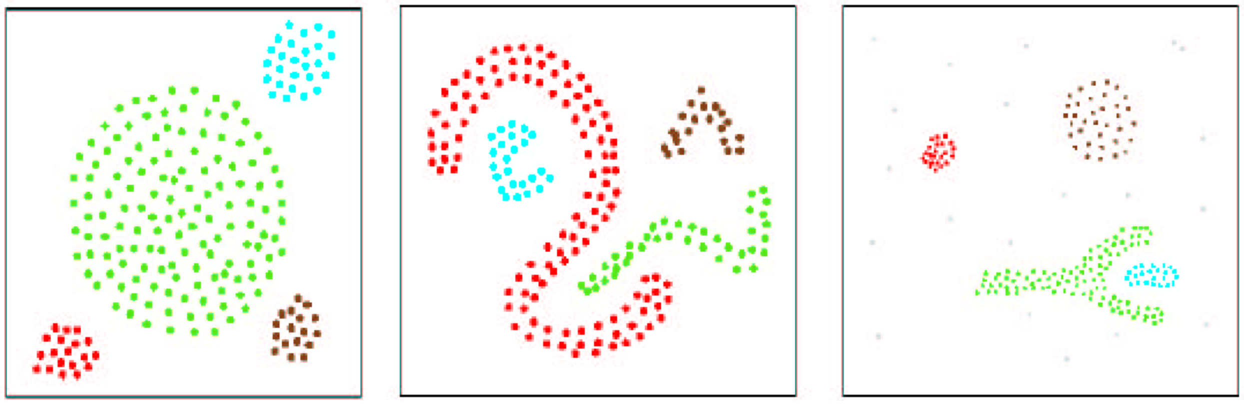
a) a set of n “objects”

b) an n x n matrix A of pairwise similarities

We get an edge-weighted graph **G** and the **GOAL** is to **partition** the vertices of the G into maximall homogeneous groups(i.e clusters)

Now since we defined the category of the clustering algorithm and what clustering does we can say the following thing. Not every cluster has to have a spherical form but we can have cluster that have unconventional form and for those types of clusters we need special algorithms that basically take the points from one space and maps it into a higher space and there, it is able to cluster the data, one example of this is the **spectral clustering** that we are going to use for the Normalized Cut.

Basically **spectral clustering** takes the degree matrix and the weighted matrix it calculates the Normalized Laplacian fully synchronized, it finds the first k smallest eigenvalues with the respective eigenvectors and it creates a matrix. It normalizes each row so each row sums up to 1 and then we apply k-means, but the eigenvalues are like a pre-processing of going from one space to another. .
Example of different types of clusters :



We have different forms of clusters as we can see in the pictures above and so we know that the k means won’t be able to clusterize the data and so for that we use other types of classifiers as the spectral clustering.
More detail about the spectral clustering when we are going to talk about the third point of the assignment.

Theory explanation behind LCA and behind the first point of the assignment

Latent Class Analysis (LCA) is a way to uncover hidden groupings in data. More specifically, it's a way to group subjects from multivariate data into "latent classes" — groups or subgroups with similar, unobservable, membership.
Above we used the 2 important terms **latent** and **class** that we are going to explain them right now:
Latent : implies that the analysis is based on an error-free **latent variable**
Class: are groups formed by uncovering hidden (latent) patterns in data.

A latent variable is a type of variable that isn't directly measurable. What happens is that the observed variables in the data act as indicators to measure the latent variables.

A better and faster way to explain LCA is that the algorithm uncovers hidden patterns of association that can exist between observations.
We used LCA for classifying data because LCA is similar to cluster analysis. The algorithm observes the data that was analyzed, then it finds connections between the data and then it groups it (the data) into clusters.

The way the model works is that within each latent class, the observed variables are independent. Well you might ask yourself why is this information so important ? Well usually the observed variables are dependent. So since the variables are independent we reintroduce the dependency by using the LCA with the following formulas :

$$p_{i_1,i_2,\ldots,i_N} \approx \sum_t^T p_t \prod_n^N p^n_{i_n,t},$$

Where T is the number of the latent classes and p_t are the unconditional probabilities and should sum to 1, instead $p^n_{i_n,t}$ are the conditional probabilities.

Now for solving the first part of the assignment we decided to use a library developed in R called **poLCA**.

poLCA which stands for **Polytomous Latent Class Analysis** is a statistical method that is used to identify unobserved groups of cases inside the data. Latent class models belong to the family of mixture models. The parameters are estimated by the EM-Algorithm. It's called EM, because it is down in two steps:

a) **„E“stimation step**

b) **„M“aximization step**

In the first one, class-membership probabilities are estimated (the first time with some starting values) and in the second step those estimates are altered to maximize the likelihood-function. Both steps are iterative and repeated until the algorithm finds the global maximum. This is the solution with the highest possible likelihood.

Sources:

- **Geeksforgeeks**
- **polCA documentation**
- **Course PDFs**

Approach that we as a team followed for the first part of the assignment + code explained

So we started by taking the dataset that has been given, we deleted the last column because it was empty and was interfering with our process. Once we did that we assigned a label to each column in such a way that after that we can manage to extract just the right features we need when we have to do the LCA algorithm.

Once we assigned the labels we divided the dataset in two parts X and Y. X contains the first 256 columns and since the values of each column are a bunch of 0 and 1 we added +1 to each cell in such a way that we won't deal with later problems when we will run the algorithm. We tried and we saw that if we use 0 and 1 then we will eventually get negative numbers during the process and this fact blocks the entire process so as the documentation suggests we added a +1 to each cell of the X set.

Y contains the last 10 columns that represent the labels. So having Y and the result of the network we can compare the result after the algorithm has finished the work.

Before running the algorithm we implemented a short function that basically takes the Y and for each row calculates the number that corresponds to and adds that number to a new matrix that we will use later for comparison.

Once we did all these steps we bound each element with the features that got labeled before, and then we ran the algorithm and we also ran the rand index as it was requested to be calculated for each k .

We run the algorithm for different values of K starting from 5 and going up to 15. The tunning that we did was based on trials, we tried until we found the right balance.

After we did all these steps we printed the results in a graphical way so we can see the number and how good the performance was.

Code:

1st step reading the data:

~~~
```{r}
library("poLCA")
# Read the dataset
dataset_readed<- read.csv(file="semeion.data", sep=" ",
stringsAsFactors=FALSE)
#Print the dataset
```
~~~

2nd step deleting the last column that was empty:

~~~
```{r}
dataset_readed <- subset( dataset_readed, select = -267 )
```
~~~

3rd step assigning the labels to the data:

~~~
```{r}
colnames(dataset_readed)<-
c("G1","G2","G3","G4",........,"G265","G266")
dataset_readed
```
~~~

4th step was separating the data into X and Y and adding the +1 to the X part of the data:

~~~
```{r}
newX<- dataset_readed[,c(0:256)]
newX <- newX +1
newY<- dataset_readed[,c(257:266)]
y<-matrix(0, nrow = 1592, ncol = 1)

```
~~~

5th step was to calculate or better to convert the number from binary to normal numbers:

~~~
```{r}
for(i in 1:1592) {
  for(j in 1:10) {
  if(newY[i,j]==1){
    y[i,1]<-j;
  }}}
```
~~~

6th step was to run the algorithm and see the results:

Reading the documentation + some stack overflow comments based on the documentation and how it works we had to tune the values a little bit until we found the right values or at least values that confirmed that we were on the right path.

~~~
```{r}
f  <- with(newX,
           cbind(G1,G2,G3,G4,G5,G6,.....,G255,G256)~1)
```
```{r}
Finalresults <- setNames(data.frame(matrix(ncol = 3, nrow =
0)), c("Model", "BIC" ," Rand Index"))
for (k in 5:15) {
  lcaResult <- poLCA(f, newX, nclass=k, maxiter=3000,
tol=1e-10, probs.start=NULL, nrep=3, verbose=TRUE,
calc.se=TRUE)

lcaResult
  model <- paste("K is uqual to = ", k, sep="")
  bic <- round(lcaResult$bic, digits=5)
  rand <- round(rand.index(lc$predclass, y), digits=5)

Finalresults[nrow(Finalresults)+1,] = c(model,bic,rand)
  print(Finalresults)
}
```
~~~

7th step: Representation of the results under a graphical way:

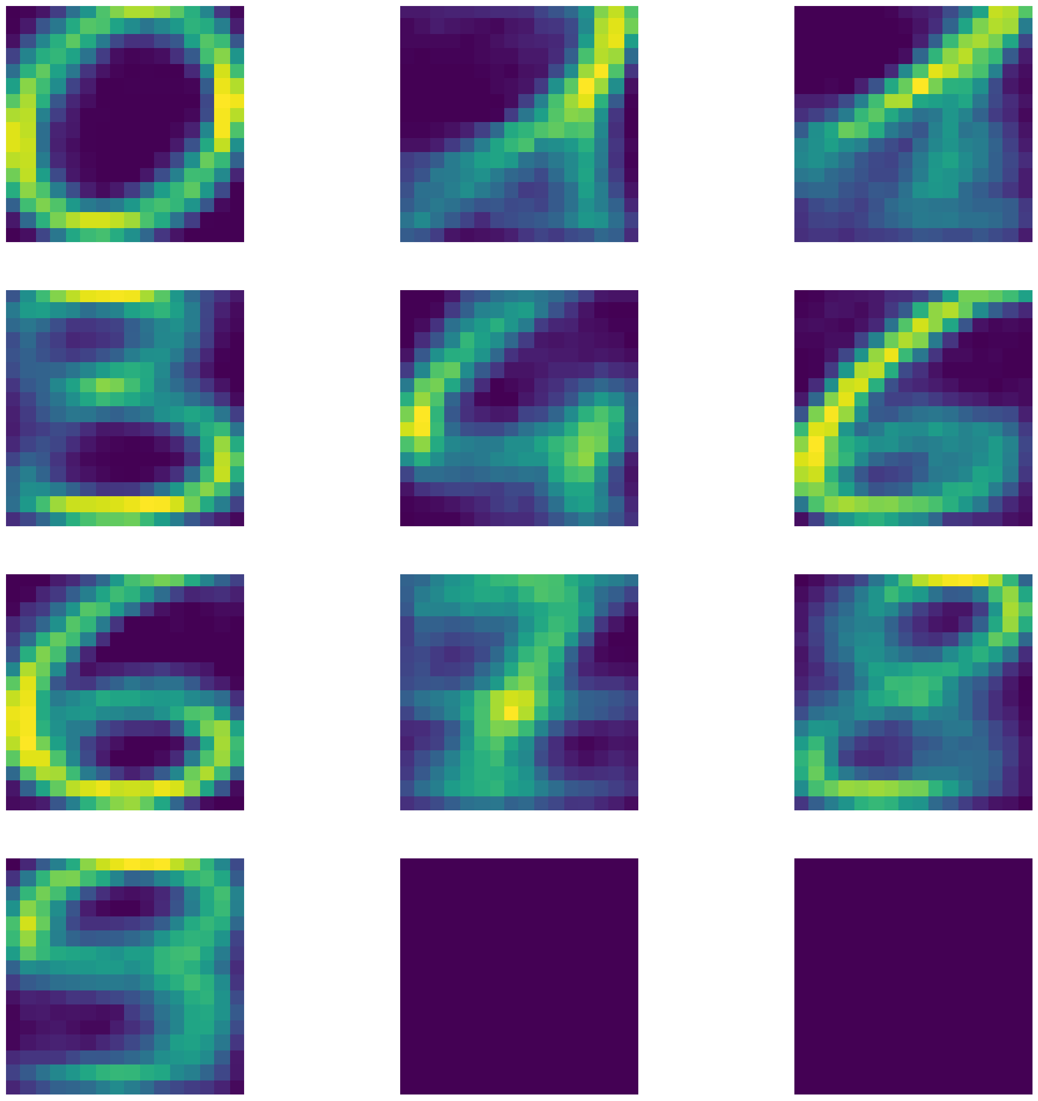
```
#probabilities of LCA model for each pixel
data = pd.read_csv("probs").drop(columns={"Unnamed: 0"})
data1 = data.iloc[:,[i for i in range(512) if i%2==0]]
data2 = data.iloc[:,[i for i in range(512) if i%2!=0]]
```

```
#real label
with open("semeion.data") as textFile:
    data = [line.split() for line in textFile]
```

```
data = np.asarray(data).astype(float)
y = data[:,-10:]
v,c = np.where(y==1)
c = c[1:]
```

```
#predicted label
predictions =
np.squeeze(pd.read_csv("predclass").drop(columns={"Unnamed:
0"}).to_numpy())
```

```
_, axarr = plt.subplots(4,3,figsize=(10,10))
label = 0
vetLabel = [9,5,8,2,4,3,0,1,6,7]
for i in range(4):
    for j in range(3):
        if label<10:

axarr[i,j].imshow(np.array(data2.iloc[vetLabel[label],:]).r
eshape(16,16))
            axarr[i,j].axis('off')
            label+=1;
        else:
            axarr[i,j].imshow(np.array([0 for i in
range(256)]).reshape(16,16))
            axarr[i,j].axis('off') #hiding the axis values
for all the 25 images
```



As we can see the number zero, one and six got a pretty good result but the numbers nine, seven , and three got poor results but for sure with some more input data the results will get better.

Theory explanation behind Mean Shift and behind the second point of the assignment:

Mean Shift belongs to the category of an unsupervised learning algorithm and the way it works is that it discovers blobs in the smooth density of the samples.

Since the mean shift is a centroid based algorithm it works by updating the candidates for centroids to be the mean of the points within a given region.

Once we do that the candidates are going to be filtered such that we can eliminate the duplicates so we can form the centroid the final centroids, so thanks to this way of processing the points / candidates we really don’t have to know the k(number of clusters) in advance.

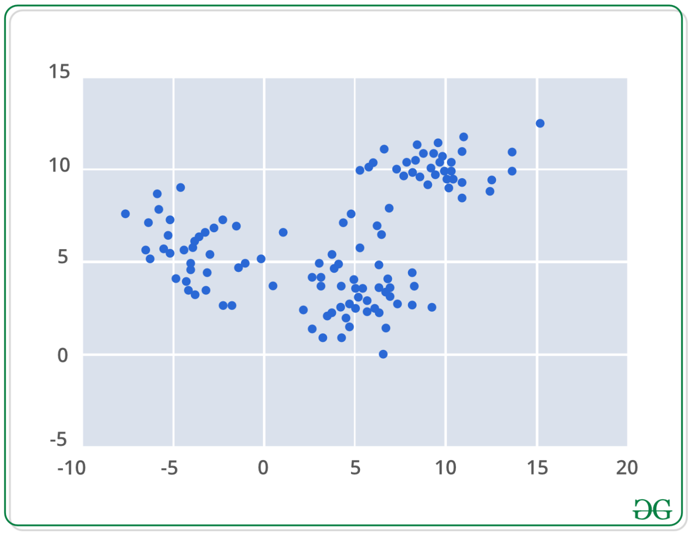
Mean-shift algorithm has applications in the field of image processing and computer vision.

Example :

Given a set of data points, the algorithm iteratively assigns each data point towards the closest cluster centroid and direction to the closest cluster centroid is determined by where most of the points nearby are at. So each iteration each data point will move closer to where the most points are at, which is or will lead to the cluster center. When the algorithm stops, each point is assigned to a cluster.

Unlike the popular K-Means cluster algorithm, mean-shift does not require specifying the number of clusters in advance. The number of clusters is determined by the algorithm with respect to the data.

The downside to Mean Shift is that it is computationally expensive O(n^2). The first step when applying mean shift clustering algorithms is representing your data in a mathematical way, by that we mean representing the data as points such as the set below.



Mean-shift builds upon the concept of kernel density estimation KDE. Imagine that the above data was sampled from a probability distribution. We used the KDE term so you might ask yourself what KDE is ? Well below you can find the explanation:

KDE is a method to estimate the underlying distribution also called the probability density function for a set of data. It works by placing a kernel on each point in the data set.

A kernel is a fancy mathematical word for a weighting function generally used in convolution. There are many different types of kernels, but the most popular one is the Gaussian kernel. Adding up all of the individual kernels generates a probability surface example density function. Depending on the kernel bandwidth parameter used, the resultant density function will vary.

Approach that we as a team followed for the second part of the assignment + code explained:

So before starting the actual algorithm we used the PCA or better we tried different values of PCA so we could see how the algorithm performs.

You might wonder what PCA is ? Well PCA helps you interpret your data, but it will not always find the important patterns.

Principal component analysis (PCA) simplifies the complexity in high-dimensional data while retaining trends and patterns. It does this by transforming the data into fewer dimensions, which act as summaries of features.

Once we did that we used two for loops one inside the other. Inside the first for loop we run the PCA and we set the values for the bandwidth allowing it to fluctuate so we can find the right value for it.

Once we set all these values we start the second for loop and we run the actual Mean Shift algorithm(we used the library of sklearn) with different values for the bandwidth and in the end we calculate the rand Index.

For the **rand index** basically we had two functions, one that we developed and the other one is from the sklearn library so we could test that what we did was right and indeed we found the same results.

After we got the results of the algorithm with the appropriate tuning we created some graphs so we could see the performance and the graphical results of the algorithm. After we ran the algorithm with the fluctuating values we ran again the algorithm with the fixed values that we found to be the right balance for a good end result and then we displayed some graphs again so we can see the performance of the best result we could get from this classifier. In the end we displayed the clusters and so we can see the numbers.

As we will see below the number zero , six and four got pretty good results instead numbers one, seven and eight got poor results but for sure with more input data we could get some better results.

Now let’s present the code:

1st step : Setting the PCA values

```
pca_components = np.array([5,10,25,50,100,200]) #we'll
tuning different PCA reduction

vect_Nclusters = []
vect_RI = []
vect_PCA = []
vect_Step = []
```

2nd step: Running the PCA + setting up the bandwidth :

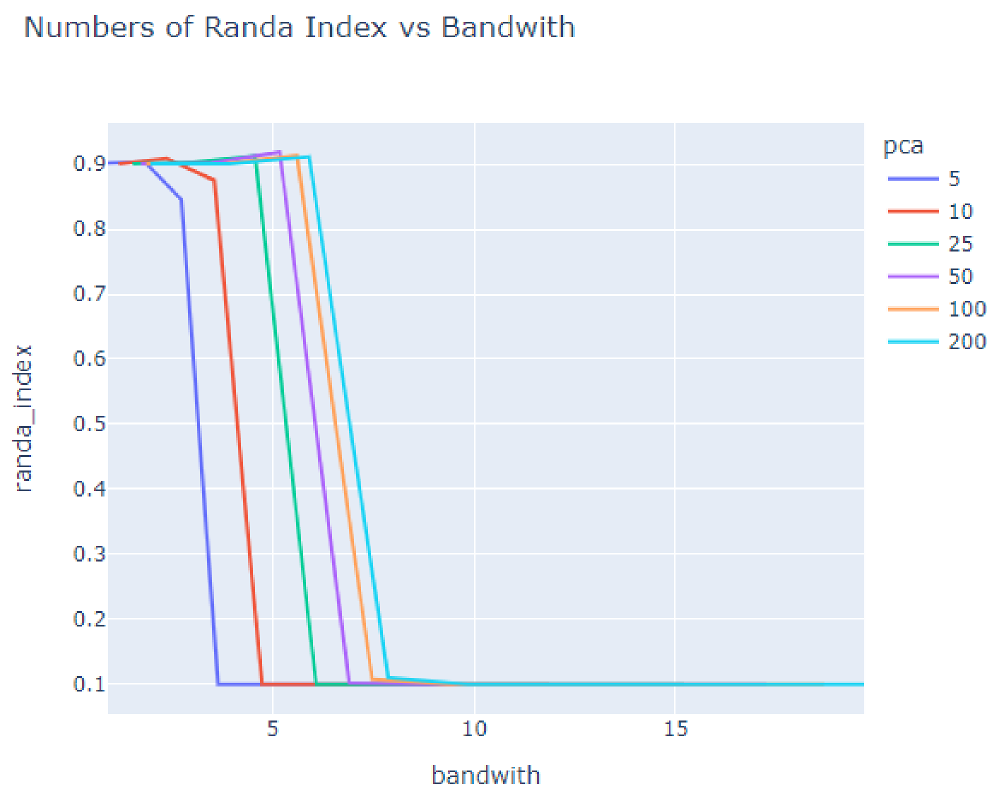
```
for components in pca_components:
    pca = PCA(n_components=components)
    X_pca = pca.fit_transform(X)
    X_pca = pd.DataFrame(X_pca)
    bandwidth = estimate_bandwidth(X_pca)
    #5 value before and after the estimated bandwith
    step = [bandwidth*((i+1)/5)for i in range(5)] +
[bandwidth+bandwidth*((i+1)/5)for i in range(5)]
```

3rd step : Running the Mean Shift algorithm + rand index

```
for i in step:
        print("pca:",components," bandwith:",i)
        ms = MeanShift(bandwidth=i) #our model
        ms = ms.fit(X_pca)

        labels = ms.labels_
        cluster_centers = ms.cluster_centers_
        labels_unique = np.unique(labels)
        n_clusters_ = len(labels_unique)
        print("number of estimated clusters :",n_clusters_)
        ri = rand_index(c,labels,1593,max(n_clusters_,10))

        print("randex_index: ",ri)
        #store informations
        vect_Nclusters += [n_clusters_]
        vect_RI += [ri]
        vect_PCA += [components]
        vect_Step += [i]
```

4th step: Graph representation of the results :

```
mean_shift =
pd.DataFrame({"clusters":vect_Nclusters,"randa_index":vect_
RI,"bandwith":vect_Step,"pca":vect_PCA})

fig = px.line(mean_shift, x="clusters", y="randa_index",
color="pca",title='Numbers of clusters vs Randa Index')
fig.show()

fig = px.line(mean_shift, x="bandwith", y="clusters",
color="pca",title='Numbers of clusters vs Bandwith')
fig.show()

fig = px.line(mean_shift, x="bandwith", y="randa_index",
color="pca",title='Numbers of Randa Index vs Bandwith')
fig.show()
```

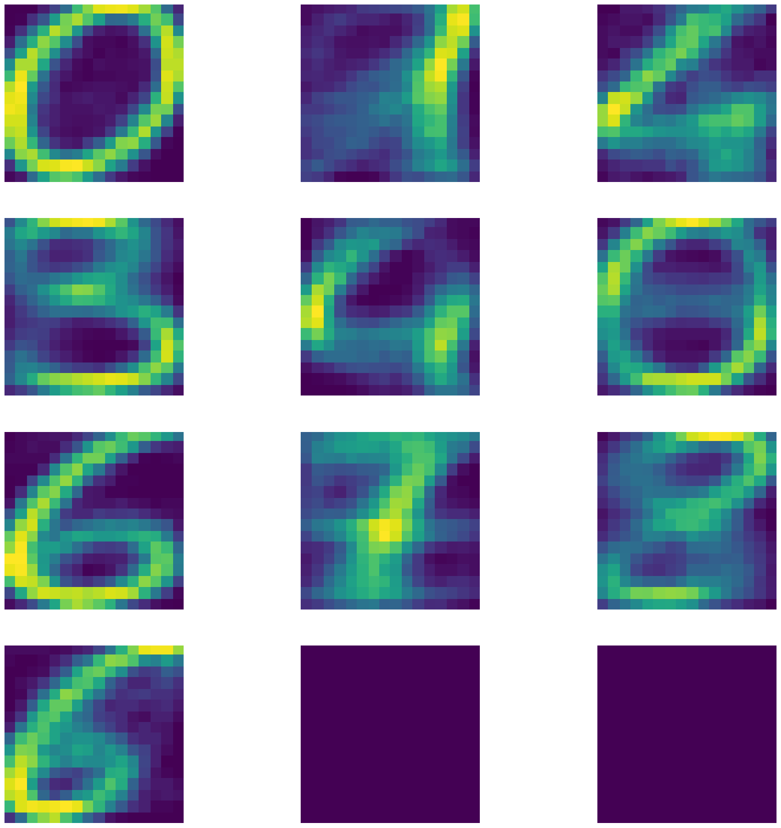


We can see that one of the best results between the bandwidth and rand index was with a value of PCA between 3 and 8.

Since the next step would be the same algorithm applied for PCA equal to 5 and a fixed bandwidth I will show directly the results since the code does not change at all.

5th step : Showing the results of the algorithm with the final tuning:

```
_, axarr = plt.subplots(4,3,figsize=(10,10))
label = 0
for i in range(4):
    for j in range(3):
        # Per iteration in the loop, picking one row 
randomly to display in our 5x5 matrix

axarr[i,j].imshow(np.sum(X[new_labels==label],axis=0).resha
pe((16,16)))
        axarr[i,j].axis('off') #hiding the axis values for
all the 25 images
        label+=1;
```



The results as we can see are pretty good for the numbers zero, six, and eight but for example for number three, one and seven for sure with some fine tuning + some more input data we could get a better result.

For more graphs we can check the code. I did not insert all the graphs because it would take entire pages taking the focus out of the important explanation of the work.

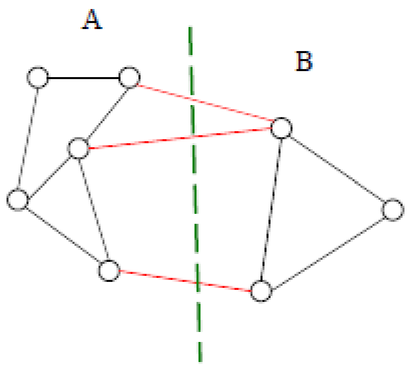
Theory explanation behind Normalized Cut and behind the third point of the assignment

Let G=(V, E, w) a weighted graph.

Given a “cut” (A, B), with B =V \ A, define:

$$cut(A, B) = \sum_{i \in A} \sum_{j \in B} w(i, j)$$

Below we can see a visual example of the problem.



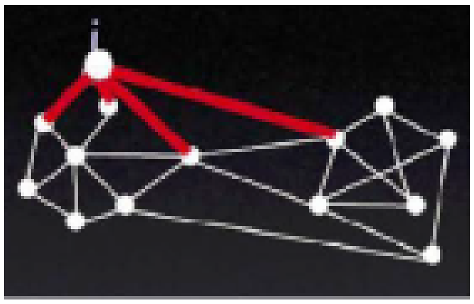
The basic idea of this problem is that we want to find the weights in that graph that minimize the cut. In doing so we found a big big problem because to find the 2 clusters and the cut we do not take into consideration what happens in between the clusters and so we will find two **unbalanced** clusters.

The good thing is that we can solve the problem in polynomial time, as for the unbalanced clusters, well, we can introduce other 2 important concepts in such a way that we could solve this important problem.
The two new concepts are :

1. **Degree of nodes :**

$$d_i = \sum_j w_{i,j}$$

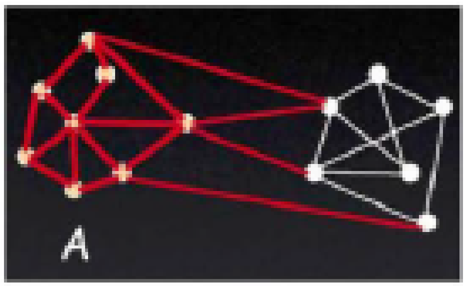
Graphical example :



2. **Volume of a set** :

$$vol(A) = \sum_{i \in A} d_i, A \subseteq V$$

Graphical example :



Now that we have introduced the two important notions we can talk about the **normalized cut** since it will solve the problem that we had before with the unbalanced class.

Normalized cut is the CUT problem that takes into consideration what happens in between the clusters. It does the CUT as before but takes into consideration the volume of A and the volume of B in such a way that now it creates the balanced clusters.

The formula below :

$$Ncut(A,B) = \boxed{cut(A,B)}\left(\frac{1}{\boxed{vol(A)}} + \frac{1}{\boxed{vol(B)}}\right)$$

So if we have the result of (1 over vol(A) and 1 over vol(B)) big then we can say that the cut is not minum but if we have them small, well at this point we can say that we multiply the cut by a small number so we can have as a result a small number. To have a small number in the end we have to have a big volume for A and B.

Below we can find the explanation of the code and of the approach.

Approach that we as a team followed for the third part of the assignment + code explained

For the last point of this assignment we decided to use spectral clustering since it allows us to clutter data that is not spherical and it is more manageable as an algorithm. (The explanation of the algorithm and how it works can be found at the beginning of the document where we explain what a cluster is)

First things we did as before we got the data from the dataset, we separated the data in X and Y.

The differences are that : now we play with the number of clusters and by that I mean that we run the algorithm multiple times with different numbers of clusters between 5 and 15 so we can see the difference in performance.

After that we try the tuning of the algorithm with different ways to assign labels: one is "**discretize**" and the other one is "**kmeans**".

Once we do that basically we make two for loops, one inside the other for being able to loop through all the combinations between the kernels and the type of different labels running obviously the algorithm with these different parameters.

Once we run the algorithm for every run we save the labels and then we run a rand index between the true labels and the labels that we found out by running the algorithm.

You might ask why using the **rand index** and what it is : **well basically** the rand index in *statistics* but also in *data clustering* is a measure of the similarity between two data clusters. We developed our own rand index function but we also used the one offered by the library that we found on sklearn.

Now let’s present the code:

1st step: Creating the range for the clusters + the range or the labels + running the algorithm and for each run calculating the rand index too:

```
n_clusters = [i for i  in range(5,16)] #tuning different cluster
numbers
assign_labels = ["discretize","kmeans"] #tuning different way to
assign labels

vect_Nclusters_ = []
vect_label = []
vect_RI_ = []

for clu in n_clusters:
    for label in assign_labels:
        clustering = SpectralClustering(n_clusters=clu, affinity =
'nearest_neighbors',n_neighbors = 10,
                     assign_labels =label, n_jobs=-1).fit(X)

        labels = clustering.labels_
        print("n_clusters= ",clu, ", ass.lab=",label," and rand
index: ",rand_index(c,labels,1593,max(clu,10)))

        vect_RI_ = vect_RI_ + [rand_index(c,labels,1593,max(clu,10))]
        vect_Nclusters_ = vect_Nclusters_ + [clu]
        vect_label = vect_label + [label]
```

2nd step: Printing the result of the algorithm:

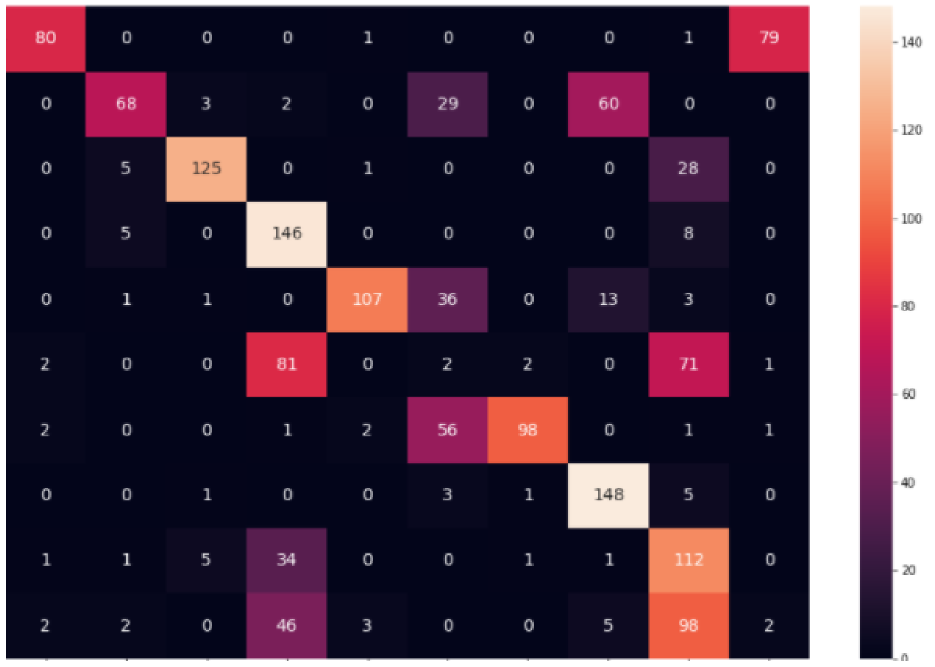
Once we did that we created a graph to show the number of clusters vs the Rand Index so we can understand how the results of the Rand Index changes based on the number of clusters since we said earlier that the number of clusters gets changed.

```
normalized_clustering =
pd.DataFrame({"clusters":vect_Nclusters_,"randa_index":vect
_RI_,"label":vect_label})
fig = px.line(normalized_clustering, x="clusters",
y="randa_index", color='label', title='Numbers of clusters
vs Randa Index')
fig.show()
```

Once we did all these steps we used a confusion matrix to see better the results and the performance.

You might ask yourself what a **confusion matrix** means.

The confusion matrix allows us to see the performance of the classifiers on a set of test data for which the true values are known.



80	0	0	0	1	0	0	0	1	79
0	68	3	2	0	29	0	60	0	0
0	5	125	0	1	0	0	0	28	0
0	5	0	146	0	0	0	0	8	0
0	1	1	0	107	36	0	13	3	0
2	0	0	81	0	2	2	0	71	1
2	0	0	1	2	56	98	0	1	1
0	0	1	0	0	3	1	148	5	0
1	1	5	34	0	0	1	1	112	0
2	2	0	46	3	0	0	5	98	2

3rd step: Running the algorithm we fix number of clusters to see just the results with this specific number of clusters.

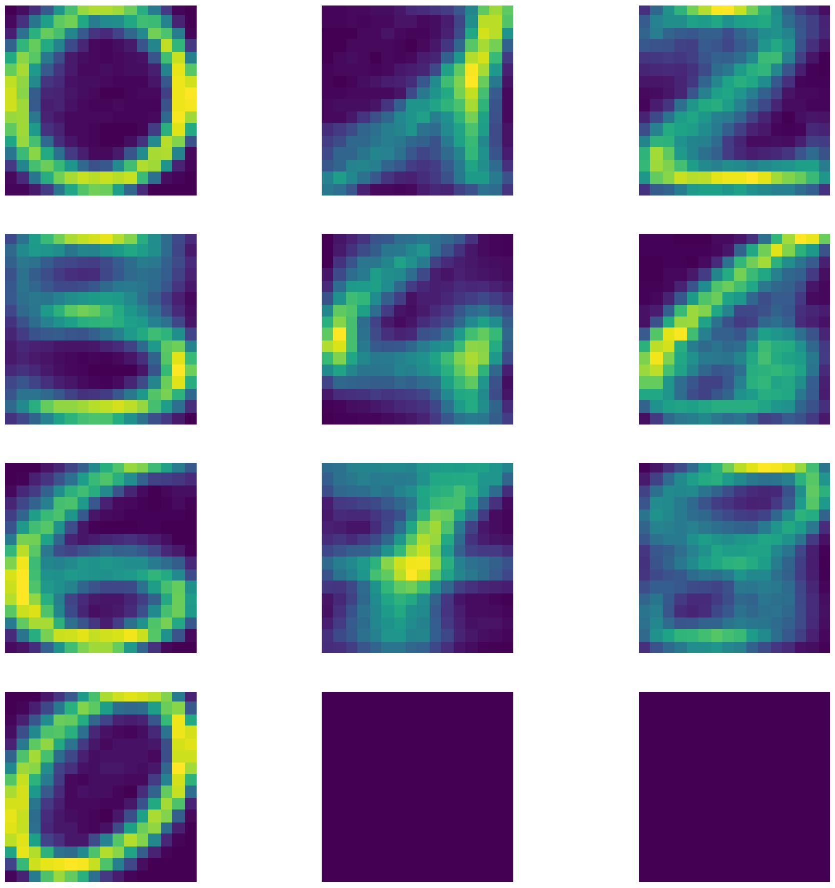
```
#let's see a particular case with 10 clusters
clustering = SpectralClustering(n_clusters=10, affinity =
'nearest_neighbors',n_neighbors = 10,
                     assign_labels ="discretize",
n_jobs=-1).fit(X)
labels = clustering.labels_
labels_unique = np.unique(labels)
n_clusters_ = len(labels_unique)
```

```
plt.figure(figsize=(15,10))
new_labels = rematchLabels(c,labels).astype(int)
sns.heatmap(confusion_matrix(c, new_labels),
annot=True,annot_kws={"size": 14}, fmt=".0f")

ri = rand_index(c,new_labels,1593,max(n_clusters_,10))
print("my randex_index: ",ri)
print("sklearn randex_index: ",rand_score(c,labels))
```

4th step: Displaying the result:

At the end of the process we run a small block of code just to see the result of the cluster and see how the algorithm performed. As we will see in the picture below the cluster performed particularly well for the number zero one two four and six, it did not perform so well on seven eight and five. Maybe with more examples it would be able to perform better.

```
_, axarr = plt.subplots(4,3,figsize=(10,10))
label = 0
for i in range(4):
    for j in range(3):
        # Per iteration in the loop, picking one row
randomly to display in our 5x5 matrix

axarr[i,j].imshow(np.sum(X[new_labels==label],axis=0).resha
pe((16,16)))
        axarr[i,j].axis('off') #hiding the axis values for
all the 25 images
        label+=1;
```



Conclusions:

Before saying anything about the performance of the 3 classifiers I would like to add a table with the results we managed to extract from the 3 classifiers and based on that we are going to say which of the 3 performed better.

On the next page you can see the table with all the values. Due to format problems I did not manage to bring the table on this page and not change the entire document format.

Now that we have the data we can clearly see that the best classifier out of the three used in this assignment is the LCA looking at the data.

In the second place we have the normalized cut which more or less got the same results as the LCA and in the third place we got Mean Shift which was the hardest to work with, because finding the right bandwidth was a real problem.

We tested for days the right bandwidth in such a way that we found the right amount of clusters and we got a decent rand index value.

The tuning of the parameters was a real challenge for the Mean Shift again because as we can see by just adjusting a value of a parameter by just a small fraction the values change quite a lot. You can see it better in the code explanation above where we showed the graphs.

Overall we manage to find the 10 clusters for each classifier but some clusters and example the clusters that represent the number zero, seven and eight are the ones that have learned the best. On the opposite part we have numbers two, three and five and are not so well defined and so the learning was not the best but not always, it changes a little bit on each classifier, for specific results on each classifier we can check the last section of each classifier above.

Certainly after this test we can clearly see the number that need more input data such that we can make the clustering better for that specific digits or maybe we still have to find a better tuning for the parameters.
After some discussions we agreed that if we would have to work again with one of the three classifiers that we used for this project and we can choose one of them, the classifier of choice we think it would be the normalized cut / spectral clustering for the theoretical proprieties and because it was so easy to understand how it works, where are the problems like the number of the clusters that you need to know in advance or the that it is sensible to outliers .

The tuning was very pleasant to do and by that we mean that reading the documentation and playing a little bit with the values we managed to find the right values pretty quick which showed us that it would be an algorithm that could help us in the future when needed since it was very reliable.

Below we can find the table with all the values

Klusters	5	6	7	8	9	10	11	12	13	14	15
LCA + BIC	443345.0899	437012.23516	432085.13294	429193.12494	424614.73711	423690.27486	421416.67331	421551.39958	419001.2233	416961.12219	423690.27486
Rand Index LCA	0.80138	0.84028	0.84724	0.86927	0.88759	0.8907	0.90052	0.89128	0.89854	0.90953	0.91902
Mean Shift Band Width	3	2.9	2.8	2.7	2.7	2.6	2.4	2.2	2.1	2.4	2.3
Rand Index Mean Shift	0.0935	0.0945	0.7635	0.8425	0.8435	0.8525	0.8535	0.8545	0.9008	0.9045	0.9065
Mean Shift PCA	5	5	5	5	5	5	5	5	5	5	5
Rand Index Normalized Cut	0.7983	0.8031	0.8739	0.8877	0.8845	0.8751	0.8805	0.8968	0.9075	0.9060	0.9007