Ca' Foscari University of Venice

ARTIFICIAL INTELLIGENCE: KNOWLEDGE REPRESENTATION AND PLANNING

[CM0472-1]

Assignment 2: Clustering

Report by:
Zonelli Mattia, 870038

Code by:
Zonelli Mattia, 870038
Garbin Eleonora, 869831

Contents

1 Introduction

The goal of the assignment was to perform classification over the Semeion Handwritten Digit data set [1] using:

- Latent Class Analysis (LCA);
- Mean Shift;
- Normalized cut.

for the mean shift and normalized cut we 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:

$$R = 2\frac{a+b}{n(n-1)}$$

where

- `n` is the number of images in the dataset.
- `a` is the number of pairs of images that represent the same digit and that are clustered together.
- `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.

2 Background

Clustering or Cluster analysis is a method used in the unsupervised learning, a type of algorithms that try to learn patterns from un-tagged data [2]. Clustering is used to group data that aren't labelled trying to find some forms of similarities between objects in the same group or cluster. Usually clustering algorithm tries to put objects that share features or with high similarity in same clusters. Similarity can be expressed in various ways: distance, affinity, density, etcc. The three algorithm, that we are going to analyze, are respectively based on distribution models, densities and graph theory.

3 Latent Class Analysis (LCA)

LCA is the clustering model that is more statistically based than the others, in fact it is based on distribution models and on the probability of classifying the cases. LCA is a technique where groups are identified and created from unobserved, or latent, subgroups, which are usually based on individual responses from multivariate categorical data. LCA models can also be referred to as finite mixtures of Bernoulli distributions [4].
Latent classes are those variables that are retrieved from the unobserved variables, they divide the cases into various dimensions according to their relation to the variables. In the cluster analysis, latent classes represent the number of clusters. The maximum likelihood method is used to compute the probability that a case fall in a latent class [3].
EM algorithm is used to estimate the local maximum likelihood of model parameters in case of presence of latent variables. It is build in 2 steps: Estimation (E) step and Maximization (M) step.

- E step: it observes data and guess the values of missing data;
- M step: the algorithm generates complete data after the E step and update the missing values of the data.

The key idea behind the EM algorithm is to use the observed data to estimate the missing ones and then updating those values of the parameters in order to maximize their likelihood.

3.1 The General EM algorithm

Given a joint distribution $p(\mathrm{X}, \mathrm{Z}|\theta)$ over observed variables X and latent variables Z, governed by parameters θ, the goal is to maximize the likelihood function $p(\mathrm{X}|\theta)$ with respect to θ.

1. Choose an initial setting for the parameters θ^{old}.
2. **E step:** evaluate $p(\mathrm{Z}|\mathrm{X}, \theta^{old})$.
3. **M step:** evaluate θ^{new} given by

   $$\theta^{new} = \underset{\theta}{\operatorname{argmax}}\, \mathcal{Q}(\theta, \theta^{old})$$

   where

   $$\mathcal{Q}(\theta, \theta^{old}) = \sum_{z} p(\mathrm{Z}|\mathrm{X}, \theta^{old}) \ln p(\mathrm{X}, \mathrm{Z}|\theta).$$

4. Check for convergence of either the log likelihood or the parameter values. If the convergence criterion is not satisfied, then let

   $$\theta^{old} \leftarrow \theta^{new}$$

   and return to step 2.

The general algorithm tries to maximize the likelihood function when there are discrete latent variables. But it can also be used when the unobserved variables are the missing values in the dataset. In this case to obtain the distribution of the observed values we need to compute the joint distribution of all the variables and then marginalize over the missing ones [4].

3.2 Mixtures of Bernoulli distributions

First of all, we need to introduce some formulas that we will later use in the final version of the EM algorithm.
Consider a set of D binary variables x_i, where $i = 1, ..., D$, each of which is governed by a Bernoulli distribution with parameter μ_i, so that

$$p(x|\mu) = \prod_{i=1}^{D} \mu_i^{x_i} (1 - \mu_i)^{(1-x_i)} \tag{1}$$

where $x = (x_1, ..., x_D)^T$ and $\mu = (\mu_1, ..., \mu_D)^T$.
If we have a data set $\mathrm{X} = \{x_N\}$, and $z = (z_1, .., z_K)^T$ that is a binary K-dimensional variable having a single component equal to 1 and all the other eqaul to 0 with $\mathrm{Z} = \{z_N\}$ we can compute the expectation of the complete -data log likelihood with respect to the posterior distribution of the latent variable as

$$\mathbb{E}_z\left[\ln p(\mathrm{X}, \mathrm{Z}|\mu, \pi)\right] = \sum_{n=1}^{N}\sum_{k=1}^{K}\gamma(z_{nk})\left\{\ln \pi_k + \sum_{i=1}^{D}\left[x_{ni}\ln \mu_{ki} + (1 - x_{ni})\ln(1 - \mu_{ki})\right]\right\} \tag{2}$$

where $\gamma(z_{nk}) = \mathbb{E}\left[z_{nk}\right]$, is the posterior probability or the so called `responsibility`, of component k given data point x_n. These responsibilities are computed using the Bayes' theorem, and they are evaluated in the E step of the EM algorithm:

$$\gamma(z_{nk}) = \frac{\pi_k p(x_n|\mu_k)}{\sum_{j=1}^{K}\pi_j p(x_n|\mu_j)} \tag{3}$$

If we consider the sum over n in (2), we see that the responsibilities enter only through two terms, which can be written as

$$N_k = \sum_{n=1}^{N}\gamma(z_{nk}) \tag{4}$$

$$\overline{x_k} = \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})x_n \tag{5}$$

where N_k is the effective number of data point associated with the component k. In the M step, the algorithm try to maximize the expcted complete-data log likelihood with respect to the parameters μ_k and π. if we set the derivative of (2) with respect to μ_k equal to zero and rearrange the terms we are going to obtain:

$$\mu_k = \overline{x_k} \tag{6}$$

We see that the mean of component k is equal to a weighted mean of the data, where the weighting coefficients are given by the responsibilities that component k takes for the data point. And the maximization of π_k is obtained doing:

$$\pi_k = \frac{N_k}{N} \tag{7}$$

which represents the intuitively reasonable result that the mixing coefficient for component k is given by the effective fraction of points in the data set explained by that component.

3.3 EM algorithm for mixtures of Bernoulli

Given a Bernoulli mixture model, the goal is to maximize the likelihood function with respect to the parameters.

1. initialize $\mu_k = \frac{1}{K}$, π_{kj} to a random values chosen from a uniform distribution in the range (0.25, 0.75) and normalize it to fulfill $\sum_j \pi_{kj} = 1$. Evaluate the initial expected log likelihood as reported in formula (2).

2. **E step**: evaluate the responsibilities with the current parameters using formula (3).

3. **M step**: re-estimate the parameters using the just computed responsibilities. So μ_k^{new} using (6) and π_k^{new} using (7).

4. Re-evaluate the new expectation of the log likelihood with formula (2) and check for convergence of it. If it does not converge, return to step 2.

3.4 Implementation and Results

Before we go any further with the implementation we have to explicit some assumptions in Latent Class Analysis:

- Non-parametric: Latent class doesn't have any assumptions on linearity, normal distribution or homogeneity;

- Data: the data should be ordinal or categorical;

- Model: the number of the equations must be greater than the number of the estimated parameter

- Conditional independence: observations should be independent in each class.

For the implementation we coded in python the just described formulas.
The EM algorithm mentioned in section 3.3 is implemented in the following function. The first two line correspond to the first and second step of the algorithm and the parameters μ_k and π_k are taken as input. Then we re estimate the parameters as said in the `M step`. The last five lines of the `for-cycle` correspond to the fourth step of the EM algorithm.

```python
def EMalg(df, pi, mu, n_iter, delta):
    loglike, responsability = logLikelihood(df, mu, pi)
    loglike_prev = loglike
    for i in range(n_iter):
        pi, mu = Mstep(df, responsability)

        loglike, responsability = logLikelihood(df, mu, pi)
        if np.abs(loglike - loglike_prev) < delta:
            break
        else:
            loglike_prev = loglike
    return pi, mu, responsability
```

where `df` is the given dataframe without the last ten columns which represent the actual digits written in the images, `pi` is π and `mu` is μ.
The expectation of the log likelihood (2) $\mathbb{E}_z\left[\ln p(\mathrm{X}, \mathrm{Z}|\mu, \pi)\right]$ is coded in function below. Inside of it there is also the operations for the E step.

```python
def logLikelihood(df, mu, pi):
n = len(df)  # rows of dataframe
k = len(mu)  # numbers of clusters
responsability = responsabilityB(df, mu, pi)  # E step
logLike = 0 # expected log likelihood
for i in range(n):
    sumK = 0
    for j in range(k):
        temp = ((mu[j] ** df[i]) * ((1 - mu[j]) ** (1 - df[i])))
        sumK += responsability[i, j] *
```

```
                    (np.log(pi[j]) + np.sum(np.log(temp.clip(min=1e-50))))
        logLike += sumK

    return logLike, responsability
```

The formula (3) coded to compute the $\gamma(z_{nk})$ that will be used in the E step

```
    def responsabilityB(df, mu, pi)
```

And the M step, where we re-estimate the parameters using the responsibilities, is implemented in the following function. Where the fourth line correspond to formula 4, the `for` construct is used to compute the formula 5, the first returned value corresponds to 7 and the second one to 6.

```
    def Mstep(df, responsability):
    n = len(df)
    k = len(responsability[0])
    d = len(df[0])
    nk = np.sum(responsability, axis=0)
    mu = np.empty((k, d))
    for i in range(k):
        gamma = (responsability[:, i]).reshape(len(df), 1)
        mu[i] = np.sum(gamma * df, axis=0) / (nk[i] + 1e-130)
    return nk / n, mu
```

K cluster	Rand Index
5	0.6091
6	0.6604
7	0.7317
8	0.8193
9	0.7896
10	0.8707
11	0.8489
12	0.8479
13	0.8418
14	0.8372
15	0.8940

Table 1: LCA rand index with different numbers of clusters

We used the Rand index to measure the accuracy of our clustering algorithm. From table (1) we see how the Rand index change with the increase of the number of clusters. Rand index does not increase linearly with the increase of the clusters but sometimes there are some drop down. Anyway we get that with an higher number of clusters also the Rand index is higher.
Then we decided to plot as grey scale images each cluster using the means μ_k of each k-cluster, μ_k is a vector of length 256 where each cell represent a pixel of the images containing the hand-written digits. So in each pixel of the images of figure (1) there is the means of that pixel (or features) between all the objects in the same cluster. In example, in position (0,0) of the top left image in the figure, there is the mean, represented in grey scale, of all the pixels in position (0,0) of all the handwritten digits in that cluster. As we expected from the behaviour of the Rand index, looking at figure (1), we notice that the more clusters we have the clearer are the digits.
Further, we can observe that with a number of cluster lower than 10 we get a quite low accuracy, while with a number of clusters greater than 10 we get a better accuracy than with 10 clusters even if the actual number of digits is ten.
From figure (1) we can also notice that there is no case where all the digits in grey scale are different, there is always some missing digit.

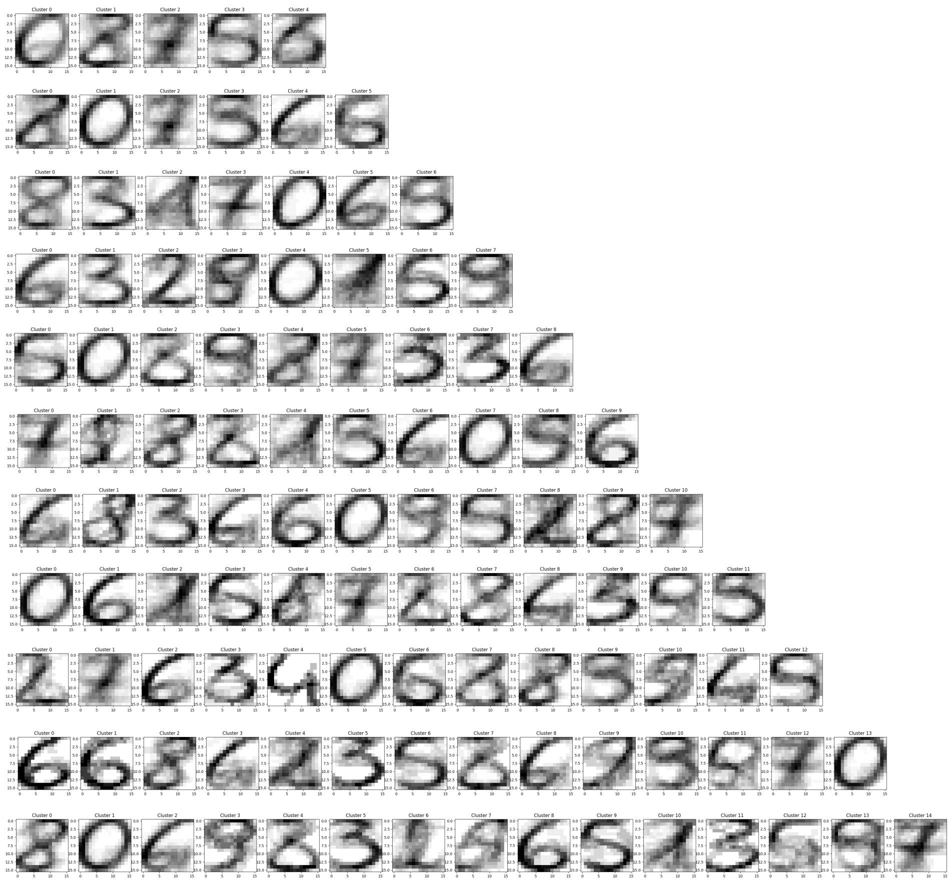

Figure 1: LCA handwritten digits in greyscale representation

4 Mean Shift

Mean shift is a density-based clustering method, an approach that aims to find clusters as areas where there is an high density of objects. When objects are in sparse areas they are considered as noise or border points.
In particular, Mean Shift is a non-parametric technique which doesn't require to know a prior the number of clusters nor the shape of the clusters, and it is a sliding-window and centroid based algorithm [6].
By not requiring the number of clusters in advance make it easier to use than the K-Means but it is way more computationally expensive. The number of clusters is established by the algorithm at execution time.
Mean shift assigns the data points to the clusters iteratively by shifting points towards the regions where there is the highest density, in fact it is considered as a Mode-seeking algorithm [5], this allow us to analyze complex multimodal feature space and to estimate the stationary points of the underlying probability density function without explicitly estimating it.

Given n data points x_i, $i = 1, ..., n$ on a d-dimensional space $\mathbb{R}^d$, with window radius h and kernel $K(x)$ defined as

$$K(x) = c_{k,d} k(||x^2||)$$

where $c_{k,d}$ is a normalizing constant that assures that $K(x)$ integrates to 1. The multivariate kernel density estimate is obtained by doing

$$f(x) = \frac{1}{nh^d} \sum_{i=1}^{N} K \frac{x - x_i}{h}$$

We have that the modes of the density function are located at the zeros of the gradient function so $\nabla f(x) = 0$.
The gradient of the density estimator

$$\begin{aligned} \nabla f(x) &= \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^{n} (x_i - x) g\left(\left|\left| \frac{x - x_i}{h} \right|\right|^2 \right) \\ &= \frac{2c_{k,d}}{nh^{d+2}} \left[\sum_{i=1}^{n} g\left(\left|\left| \frac{x - x_i}{h} \right|\right|^2 \right) \right] \left[\frac{\sum_{i=1}^{n} x_i g\left(\left|\left| \frac{x - x_i}{h} \right|\right|^2 \right)}{\sum_{i=1}^{n} g\left(\left|\left| \frac{x - x_i}{h} \right|\right|^2 \right)} - x \right] \end{aligned}$$

where $g(s) = -k^{'}(s)$. The first term is proportional to the density estimate at x computed with kernel $G(x) = c_{k,d} g\left(||x||^2\right)$ and the second term represents the mean shift

$$m_h(x) = \frac{\sum_{i=1}^{n} x_i g\left(\left|\left|\frac{x-x_i}{h}\right|\right|^2\right)}{\sum_{i=1}^{n} g\left(\left|\left|\frac{x-x_i}{h}\right|\right|^2\right)} - x.$$

The mean shift procedure obtained by alternating the computation of the mean shift vector $m_h(x^t)$ and the translation of the sliding window $x^{t+1} = x^t + m_h(x^t)$ is guaranteed to converge to a point where the increase in the density is maximized and the gradient of density function is equal to zero.
And so, we can summarize the Mean shift algorithm as follow:

1. Choose a search window size.
2. Choose the initial position of the search window.
3. Compute the mean shift vector and find the mean location inside the search window.
4. Move the center of the search window to the mean location computed in step 3.
5. Repeat steps 3 and 4 until the gradient of density function is zero.

4.1 Implementation and Results

Because it is not guarantee that Mean shift will converge in a high dimensional space like in our case, we need to apply Principal Component Analysis (PCA) to our dataset before actually running the mean shift algorithm [7].
PCA is a technique that attempts to map an high-dimensional features space to lower dimensional space while preserving the variance. Obviously, reducing the dimension of the space, we lose accuracy, but we gain simplicity in manipulating the data and in our case the confidence that the mean shift algorithm will converge [8] [9].
For both PCA and Mean Shift we used the functions already implemented by `sklearn`. In particular to apply PCA on the data

```
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.decomposition import PCA
```

```
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(features)
    pca = PCA(n_components=2)
    X_principal = pca.fit_transform(X_scaled)
    X_principal = pd.DataFrame(X_principal)
```

where `features` is an ndarray of as many row as the handwritten digits and 256 columns that correspond the the pixels of each 16x16 image.
While for the Mean Shift algorithm

```
    ms = MeanShift(bandwidth=bandwidth)
    ms.fit(X_principal)
    labels = ms.labels_
```

where `bandwidth` is changed in the interval [2.1, 3.1] and `labels` is an array containing for each images the label of its cluster. The number of clusters is established in the execution of the instruction `ms.fit(X_principal)` so to know it we need to run the following line:

```
    n_clusters_ = len(np.unique(labels))
```

In the figure 2 we decided to plot the variance in relation to the number of components used in the PCA. A higher percentage of variance means that we retain more variance from the original data which corresponds to a smaller loss of information. From this figure we can understand that the lower the number of components the lower the loss will be. In fact there is a significant drop of variance when we go from 3 to 4 PCA's features. And so, a possible choice could be to use the first 2 features.

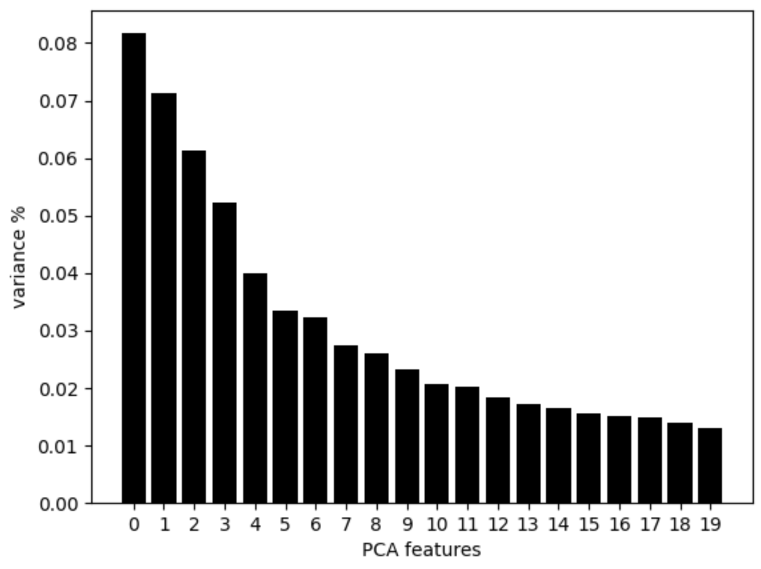

Figure 2: Variance precision in relation to PCA number of components

Hence we compared cases where we have 2, 4, 6 and 10 components. Obviously with different number of components but same bandwidth range we got different number of cluster. Figure 3 shows that for all the cases at the increase of the bandwidth correspond an increase of the number of clusters and shows how there is a huge increase on the number of cluster as we increase the number of components. In particular the number of clusters rise a lot in correspondence of the drop in the variance that we have seen in figure 2. Indeed, as we have a lower variance from 4 PCA features and above, the rise in the number of clusters is significantly large.

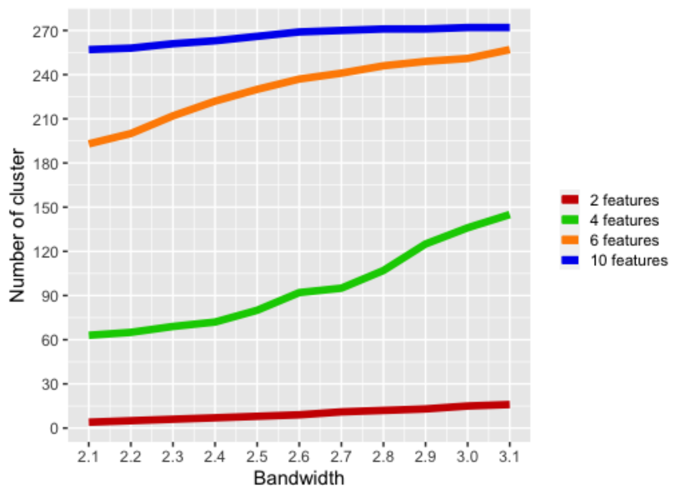

Figure 3: Number of clusters in relation to the Kernel width for different number of components in PCA.

Then we wanted to see the behaviour of Rand index for different number of components in PCA.
And so in figure 4 (a) we kept the bandwidth the same for all the cases and we plotted the according Rand index for different number of components in the PCA. From the picture we can see that with 2 PCA features the rand index increase according to the bandwidth. While for the other three cases the rand index is quite high but its increase is almost zero. Although we have to remember that for the kernel width in range [2.1, 3.1] when we have more than 2 components the number of cluster is way bigger thus with many more groups it is easier for each centroid to cover a smaller area and therefore the density is higher and consequently also the rand index.
Instead, in the right hand side (b) of figure 4, for each case we changed the range of the bandwidth in order to have more or less the same range of numbers of clusters. From this plot we notice two things. First that with a number of cluster greater than 15 there is almost no increase in the Rand index, so we can limit our comparison in the range 5 to 15 clusters. Then, we can observe that having only 2 components in the PCA it is not worse than having more. In fact when the number of cluster is less than 10, the case of 2 features is roughly better then the other cases. And even when we

have more clusters, the red line is not too lower than the others.
After seeing this plots, in order to preserve as much information as possible, we decided to continue our analysis with two components in the PCA which still guarantees a fairly high Rand Index.

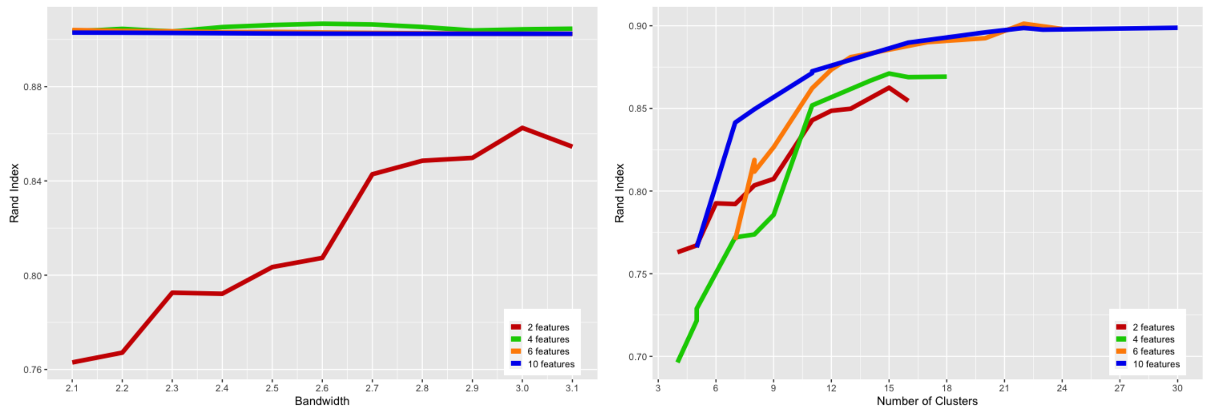

Figure 4: In the left hand side (a) we have the variation of Rand index according to the bandwidth for different number of PCA features. In the right hand side (b) we have the variation of Rand index according to the number for cluster for different number of PCA features.

Bandwidth	K cluster	Rand Index
3.1	6	0.7745
3.0	7	0.7942
2.9	8	0.8179
2.8	10	0.8267
2.7	10	0.8127
2.6	11	0.8183
2.5	12	0.8141
2.4	13	0.8199
2.3	16	0.8534
2.2	18	0.8624
2.1	21	0.8620

Table 2: Mean Shift bandwidth with relative numbers of cluster and rand index with 2 PCA components

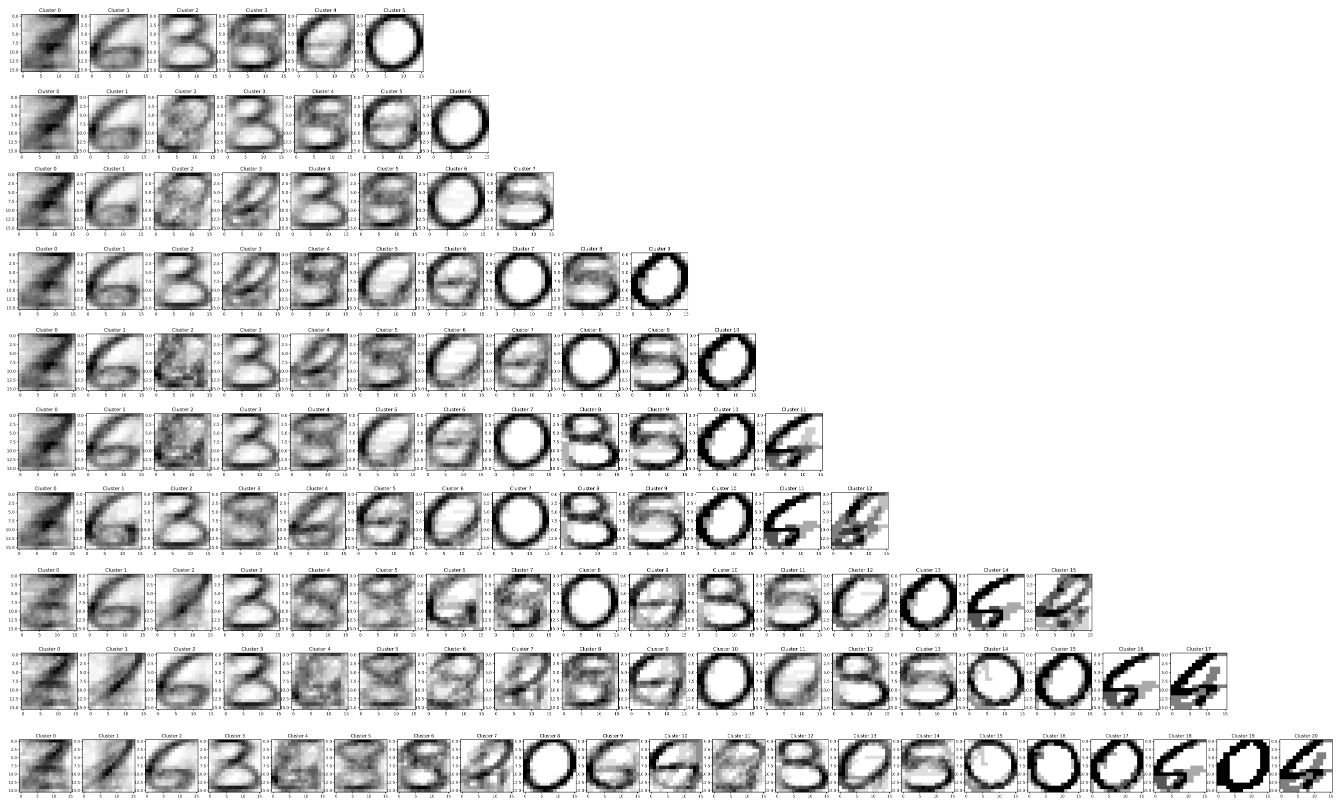

Figure 5: Mean shift, grey-scale representation of handwritten digits

---

All the images in the figure 5 are derived as mentioned for the LCA algorithm and in each row of the figure there is the result obtained running the algorithm with a different number of clusters, first row is with 5 clusters and last row is with 15 clusters. We see that with more clusters we get also much more clearer images. However there is some images that is shared between the rows but a bit less blurred in the lower rows. This could happen because with the increase of the number of clusters, some points, that before could have been outliers for a cluster, are now clustered separately, reducing so the noise and blurring.
Furthermore, in the figure 6, we can see what i have just described. Some point that in the top left plot was colored in khaki become yellow or lime in the top right plot. We can see this thing happening also with the other two

plots of the same figure. This can be explained because with more cluster centers (centroids) each of them can move to a more dense position and points that were outliers become regular points.

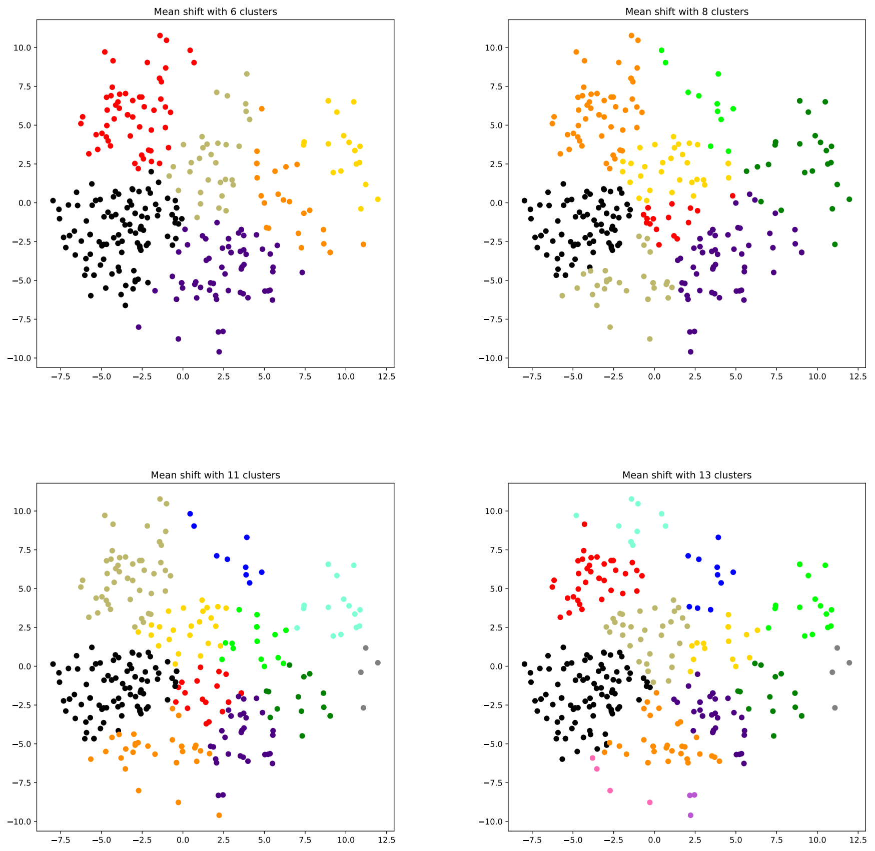

Figure 6: Mean shift datapoints with clusters in different colours

5 Normalized Cut

Before talking about Normalized cut and the version we are going to use, we first need to briefly introduce the K-means algorithm.

5.1 K-Means

K-Means algorithm is a centroid-based algorithm with the goal to partition a set of objects in K clusters, where each cluster is identified by a centroid, and each object belong to the cluster of the nearest centroid. The number of cluster have to be decided in advance. To guarantee its convergence we need to ensure that every cluster has at least one data point. Possible techniques for doing this include supplying empty clusters with a point chosen at random from points far from their cluster centers.
It can be easily implemented in a iterative way:

- **Initialize:** Pick K random points as cluster centers (centroids);
- **Alternate:**
  1. Assign data points to closest centroid;
  2. Change the cluster center to the average of its assigned points.
- **Stop:** when no points' assignments change.

K-means try to minimize the distances between points intra-cluster and maximize distance of points in different clusters. However, it converges only to a local minimum of the error function and its is very sensitive to outliers and to initialization.

5.2 Images as Graph

An arbitrary feature space of a set of points can be seen as a weighted undirected graph $G = (V, E)$ where each node in V is a point in the feature space and each edge $e_{i,j} \in E$ has a weight $w_{i,j}$ that represent by a sort of similarity between the two nodes i and j. In our case, we can represent an image as a graph if we represent pixels as nodes of the graph, there is an edge between every pair of pixels and each edge is weighted by the similarity (affinity) of the two nodes. Therefore, the clustering is like to cut up this graph to get sub-graphs with strong interior links. To retrieve the weights of the edges we can get the similarities from the distances. If we suppose to represent each pixel by a feature vector x, and define a distance function appropriate for this feature representation. Then we can convert the distance between two feature vectors into a similarity (affinity) with the help of a Gaussian kernel:

$$exp\left(-\frac{1}{2\sigma^2}dist(x_i, x_j)^2\right),$$

where the σ is the scale, and it affect the affinity. In fact a small σ will group only nearby point and large σ will group far-away point [10].

5.3 Normalized Cut

Let $G = (V, E, w)$ be a weighted graph. Let A and B be two subsets of vertices. First we need to define
the degree of the nodes as:

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

the volume of a set as

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

a "cut" (A,B), with $B = V \setminus A$ as

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

And a normalized cut as

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

In the normalized cuts approach, for any cut$(S, \overline{S})$ in G, ncut$(S, \overline{S})$ measures the similarity between different parts [11].

Let D be an $n \times n$ diagonal matrix with d on the diagonal, and let W

be an $n \times n$ symmetric matrix with $w_{i,j} = w_{j,i}$.
Thereby we can formulate the normalized cut as a minimization problem:

$$\min_{(S,\overline{S})} ncut(S,\overline{S}) = \min_{y} \frac{y^T(D-W)y}{y^tDy} \tag{8}$$

subject to:

- $y_i \in \{1, -b\}$, for some constant $-b$
- $y^T D1 = 0$

Unfortunately, this problem is NP-hard. So to make it solvable in a finite time, we need to relax the constraint that y be a discrete-valued vector and allow it to take on real values. Hence the left side of equation (8) is equivalent to:

$$min_y \; y'(D-W)y \qquad s.t. \; y'Dy = 1$$

This amounts to solving a generalized eigenvalue problem:

$$(D-W)y = \lambda Dy$$

for the second smallest eigenvalue. We look for the second smallest eigenvalue because the smallest one of Laplacian is always 0 and it correspond to the partiotion $A = V$ and $B = \{\}$.

Before moving forward, we need to define the unnormalized graph Laplacian as

$$L = D - W$$

and the Symmetric normalized graph Laplacian as

$$L_{sym} = D^{-\frac{1}{2}}LD^{-\frac{1}{2}} = I - D^{-1}WD^{-\frac{1}{2}}$$

Now, we can implement the Ncut algorithm with 2 approches.

Recursive approach of Ncut with more than 2 clusters:

1. Given a weighted graph $G = (V, E, w)$, summarize the information into matrices W and D.
2. Solve $(D - w)y = \lambda Dy$ for eigenvectors with the smallest eigenvalues.

3. Use the eigenvector with the second smallest eigenvalue to bi-partition the graph by finding the splitting point such that Ncut is minimized.
4. Decide if the current partition should be subdivided by checking the stability of the cut, and make sure Ncut is below the pre-specified value.
5. Recursively re-partition the segmented parts if necessary.

But this approach is computationally wasteful and it only uses the second eigenvector while also the other contain partitioning information.

Iterative approach using k eigenvector for more than 2 clusters:

1. Construct a similarity grap and compute the unnormalized graph Laplacian L.
2. Compute the k smallest generalized eigenvectors $u_1, u_2, ...u_k$ of the generalized eigenproblem $Lu = \lambda Du$.
3. Let $U = [u_1 \ u_2 \ ... \ u_k] \in \mathbb{R}^{n \times k}$.
4. Let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U.
$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1k} \\ u_{21} & u_{22} & \cdots & u_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n1} & u_{n2} & \cdots & u_{nk} \end{bmatrix} = \begin{bmatrix} y_1^T \\ y_2^T \\ \vdots \\ y_n^T \end{bmatrix}$$
5. Thinking of y_i's as points in $\mathbb{R}^k$, cluster them with k-means algorithm.

5.4 Spectral Clustering

However, we can find in polynomial time a cut$(S, \overline{S})$ of small normalized weight ncut$(S, \overline{S})$ using spectral techniques.
Spectral Clustering takes advantages of some properties of the graph Laplacian L

- L is symmetric and positive semi-definite:
$$f'Lf = 1/2 \sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2 \geq 0$$
for all vectors f in $\mathbb{R}^n$.

- the smallest eigenvalue of L is 0 and the corresponding eigenvectors is $\mathbb{1}$.
- This eigenvalues are $0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$.

Thus, with some small changes from the iterative version of the normalized cut, the Spectral Clustering can be implemented as follow [12]:

1. Construct a similarity graph and compute the normalized graph Laplacian L_{sym}.
2. Compute the k smallest eigenvectors $u_1, u_2, ...u_k$ of L_{sym}.
3. Let $U = [u_1 \; u_2 \; ... \; u_k] \in \mathbb{R}^{n \times k}$
4. Normalize the row of U to norm 1.
$$U_{ij} \leftarrow \frac{U_{ij}}{\left(\sum_k U_{ik}^2\right)^{1/2}}$$
5. Let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U.
6. Thinking of y_i's as points in $\mathbb{R}^k$, cluster them with k-means algorithm.

Some advantages of the this algorithm are that it is able to find clusters with non-convex boundaries.

5.5 Implementation and Results

For the implementation of Spectral Clustering algorithm we used the functions provided by `sklearn`. Spectral Clustering use the K-means algorithm so we need to decide in advance the number of clusters, hence why we need the parameter `k_clusters` in the function below. Another parameter we needed to specify was the method to go from our data to the affinity matrix, we opted to use `nearest neighbors` algorithm.

```
from sklearn.cluster import SpectralClustering
def spectral_clustering(features, k_clusters):
    X = features

    spectral_clusters = SpectralClustering(n_clusters=k_clusters,
                        assign_labels='kmeans', random_state=0,
                        affinity='nearest_neighbors', eigen_solver='amg')
    labels = spectral_clusters.fit_predict(X)

    return labels
```

K cluster	Rand Index
5	0.8141
6	0.8183
7	0.8336
8	0.8588
9	0.8771
10	0.8857
11	0.8980
12	0.8942
13	0.9007
14	0.9022
15	0.8986

Table 3: Spectral clustering, number of cluster with the according Rand Index

As we can see from table 3 the Rand Index increase with the raise of the number of cluster. The results can change a bit at each execution, anyway we always get the maximum accuracy when we have more than 10 clusters. This may happen because with too few clusters, the algorithm have to put in the same group different digits.
With the help of figure 7 we can confirm what i have just described, in the rows whose represent results of clustering with an higher number of clusters we see less blurred digits. Obviously when we have more than 10 clusters, some digits is repeated. And in the case where we have exactly 10 cluster we didn't get exactly ten differn digits.

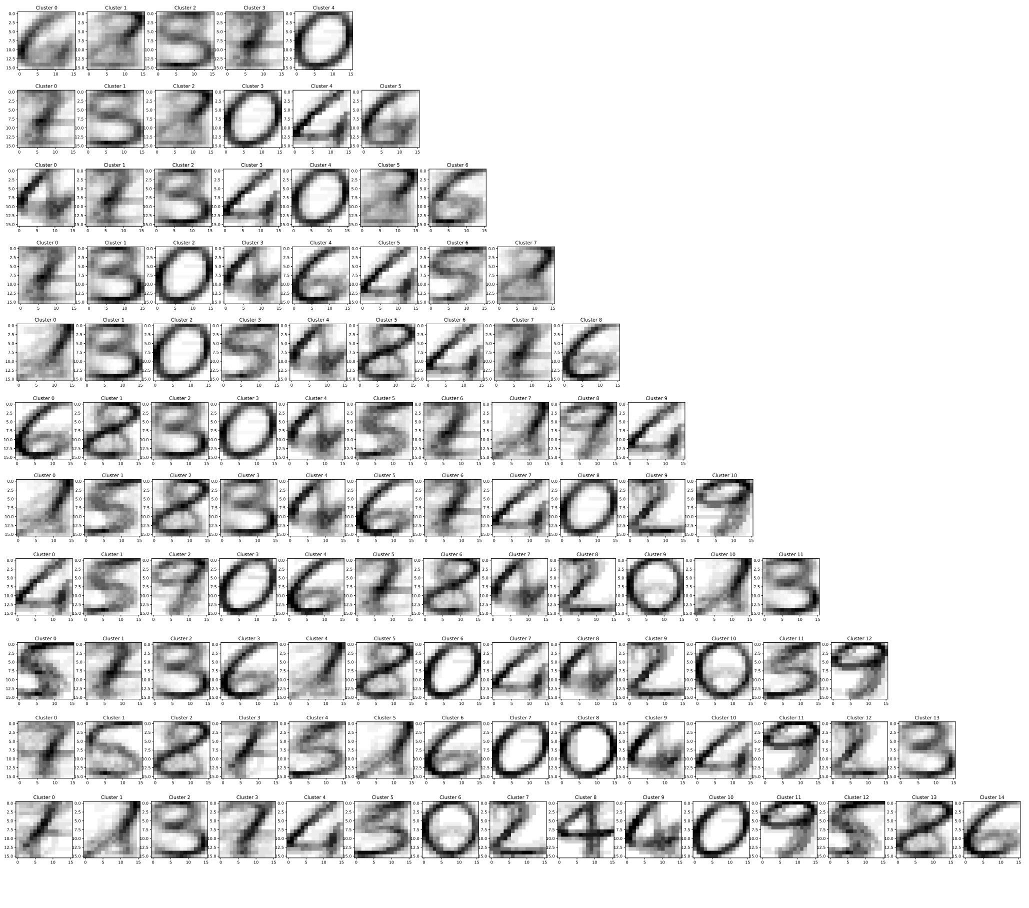

Figure 7: Spectral Clustering, grey-scale representation of handwritten digits

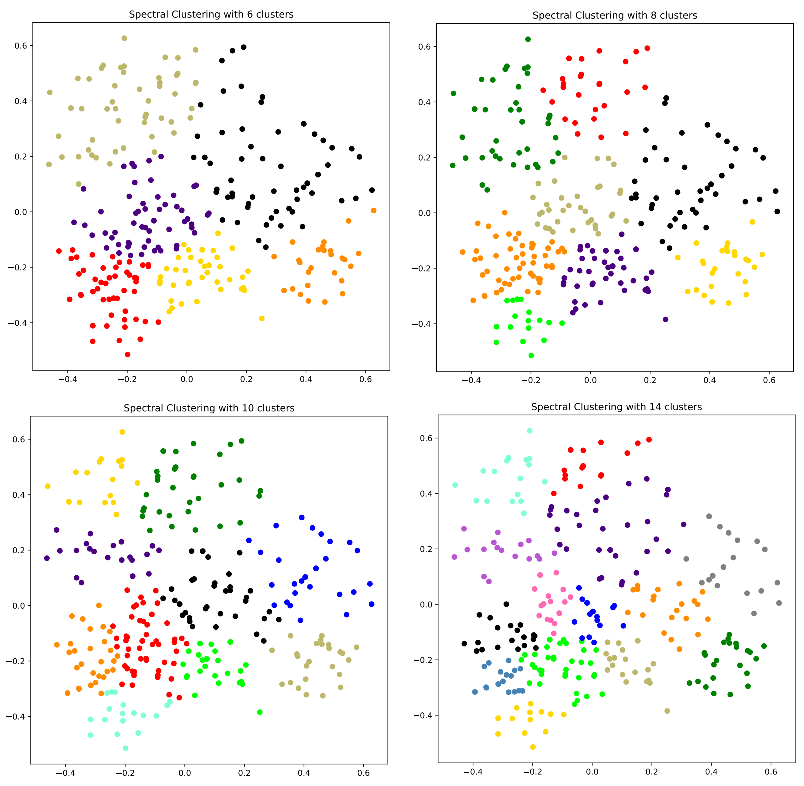

Figure 8: Spectral Clustering datapoints with clusters in different colours

From the figure 8 we can notice that with the increment of the number of clusters, some points that before can be consider as an outliers become part of the new clusters. Therefore, we get an increase of accuracy and an higher Rand Index. But i think that with too many cluster probably some point that should be labelled together could be divided.

6 Comparison and Conclusion

The goal of the assignment is to perform a classification algorithm on a given dataset of images of handwritten digits. Each row correspond to an 16x16 image and has length 256 which are the pixel of the corresponding image. Each pixel is represented by a 0 or 1 if it is white or black, this allow us to use the data as probability distribution or to assume that the images are vectors in a 256 dimensional Euclidean space. Obviously, the algorithms should group together images containing the same or similar digits.
The first difference that we can find between these methods is how they handle data. In Mean Shift and Normalized cut we can assume that images are vectors in in a 256 dimensional Euclidean space, while for LCA the images are considered as binary variables governed by a Bernoulli distribution. This happen because all of them are based on different methods to establish the clusters.
As we've seen, LCA is more probability-oriented which use the EM algorithm to estimate the maximum likelihood to estimate the probability that an observations fall in a latent class, which are the cluster. For sure, being based on probabilities make it more flexible and even more robust, in particular, if we have missing values or categorical data. Otherwise, without missing data and with ordinal variables, the other two algorithms could perform better.
Both Mean Shift and Spectral clustering can be viewed as variants of the K-Means. The differences from the K-Means are that Mean Shift tries to find the number of cluster by itself and Spectral clustering use graph-theory to find similarities between data and only at the end use the K-means to divide them into clusters.
Mean shift is a centroid based algorithm where we do not have the restriction to decide in advance the number of clusters but it requires the bandwidth of the kernel, in other words the windows size, which will determines the number of clusters. So, according to the width of the kernels, Mean shift will assign a certain number of centroids to the space and each of these will move toward densest regions. Due to the shift operations of the cluster centers, the algorithm is pretty computationally intensive. Moreover it is also very sensitive to outliers whose can be clustered together with the wanted major clusters and causing a displacement of the centroids in an area less dense than desired. Having only the bandwidth parameter makes it really scalable.
Instead, Spectral clustering can be thought as a graph clustering. Hence an important step is to pick the right method to compute the affinity between

points and to transform the data from the original space to a Laplacian graph. Regardless it use the K-means so we need to guess the number of clusters in advance. Therefore the result still really depends on the choice of the number of cluster which can bring the algorithm to include some noise in some clusters. But it is slightly more stable than K-Means because of the transformation applied to data points.
The first analysis is to compare the Rand Index between all of the algorithms.

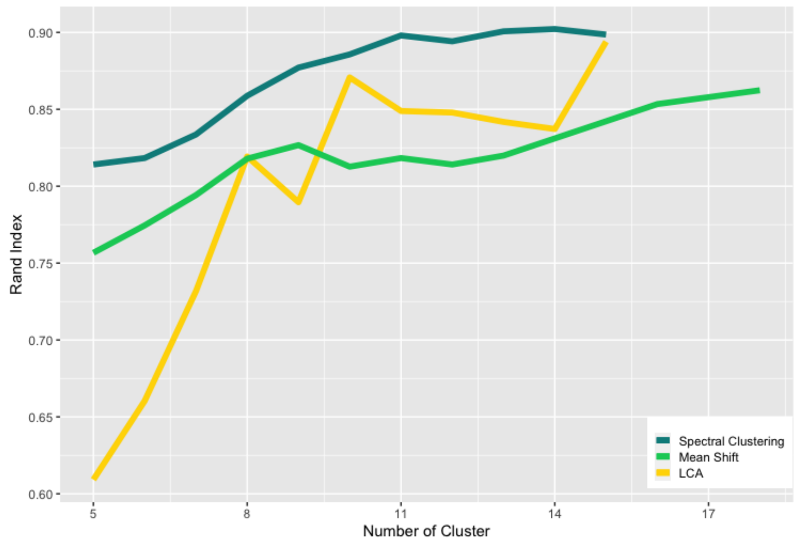

Figure 9: Number of Cluster with according Rand Index for the three algorithms.

From figure 9 appears that for LCA the increase of the number of cluster affects a lot the rise of Rand Index but there are plenty of oscillations. Even if Mean Shift starts with a higher Rand Index, LCA, with more than 10 clusters, brings better results. Then we can observe that, with 15 clusters, LCA and Spectral Clustering have more or less the same accuracy but the second one has a more linear increment and stabilize at about 11 cluster. Despite Mean Shift tries also with more than 15 clusters, it does not reach the same level of accuracy of the other algorithms.

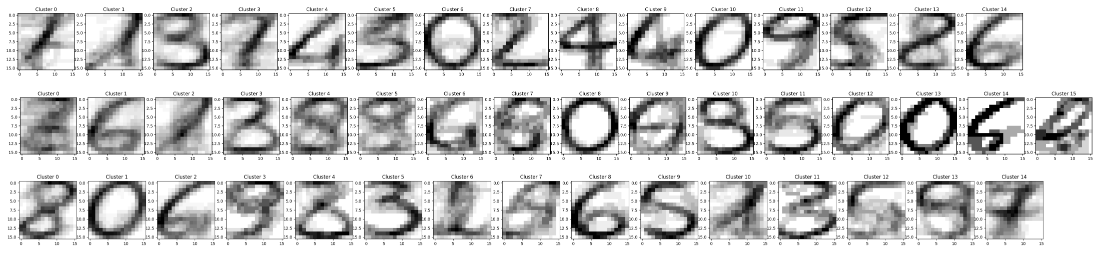

Figure 10: Handwritten digits in grey-scale with 15 clusters for all the algorithms. Top row is the result of Spectral Clustering. Middle row is the result of Mean Shift. Bottom Row is the result of LCA.

Figure 10 confirms what we have seen in figure 9, Spectral clustering and LCA have an higher precision than Mean shift. Actually a lot of the digits in the top and bottom rows of figure 10 are way more clearer than the digits in the middle row.
Moreover we have seen that Mean Shift is quite sensible to noise so the bad performance can be caused by the presence of outliers or quite sparse observations.

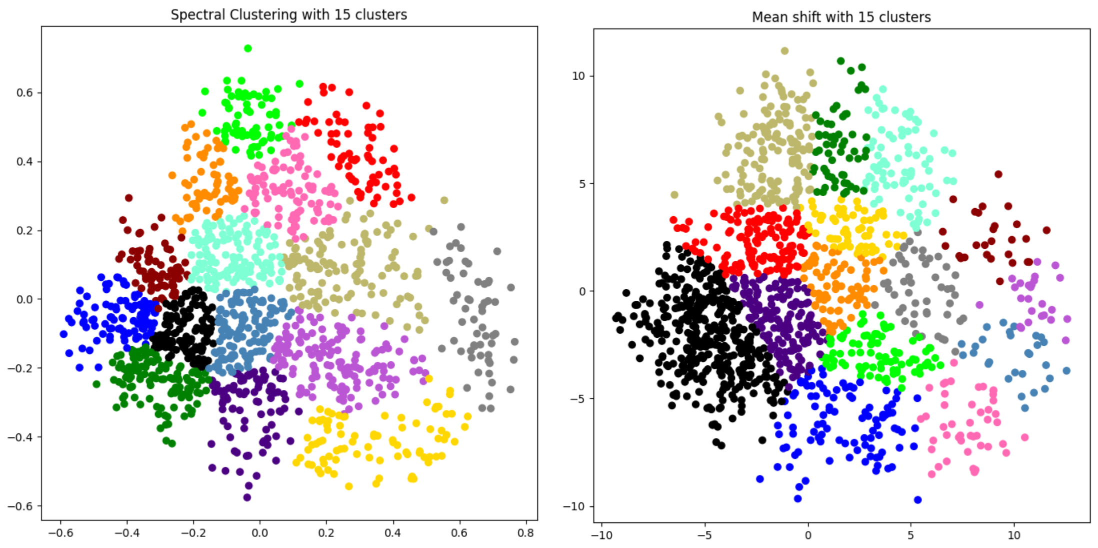

Figure 11: Comparison between Spectra clustering result (in left hand side) and Mean shift (right hand side) result with 15 clusters.

Probably outliers haven't really affected the other two algorithms because LCA with the re-estimation of parameters makes it very stable against noise. Instead, Spectral Clustering, being able to find cluster with non-convex boundaries was less affected by sparse points than Mean Shift.
In the figure 11, we see that the results of the two algorithm, Spectral Clustering and Mean shift, are pretty different. In the image in the left hand side, Spectral Clustering grouped together wider or more elongated clusters, where point are very close to each other. On the other side, resulting clusters of Mean Shift are a bit more convex and compact, but a few of them are not very dense. This could be a possible reason why the Mean shift performed a bit worse than the other two analyzed algorithms.

To sum up, after performing three different clustering algorithm on handwritten digit images we found out that:

- LCA, the method based on probability, performed quite well, being very stable against noise, but the number of clusters has a big impact on its Rand Index and so on its accuracy.

- Mean shift, which is based on density estimation, performed not too bad even with a small number of clusters but its Rand Index does not increase a lot with the increase of the number of clusters as happens to the other algorithms. A big advantage is that it doesn't need the number of clusters as parameter, but estimates it by itself, unfortunately this is computationally very expensive. Probably because the data where fairly sparse, its best Rand index is not as good as the other two.

- Spectral Clustering, based on graph theory, compared to the other algorithms requires more parameters to be executed and to find the right configuration we needed to tune them a bit. The capability to find non-convex clusters makes it a bit more versatile than other two, but as a big drawback has that we need to give to it the number of clusters in advance. On the average, it had the highest Rand Index for all values of number of clusters.

Therefore, all of them use different metrics to cluster the data. So if we have to choose one algorithm, we need to know if we are aware of the number of clusters, if not, we go with Mean Shift. If we want a more stable algorithm against outliers, we'll probably go with LCA. And if we are interested in discovering elongated and non-convex clusters, we'll choose Spectral Clustering.

References

[1] Semeion Handwritten Digit Data Set: http://archive.ics.uci.edu/ml/datasets/Semeion+Handwritten+Digit

[2] Unsupervised Learning: https://en.wikipedia.org/wiki/Unsupervised_learning

[3] Latent Class Analysis: https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/latent-class-analysis/

[4] C. Bishop, Pattern Recognition and Machine Learning, 2006

[5] Mean Shift: https://www.geeksforgeeks.org/ml-mean-shift-clustering/

[6] Mean shift clustering: https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/TUZEI

[7] Mean shift sufficient condition for convergence: https://www.sciencedirect.com/science/article/pii/S0047259X14002644?via%3Dihub

[8] PCA for Clustering: https://rpubs.com/cyobero/pca-clustering

[9] Step by Step PCA: https://builtin.com/data-science/step-step-explanation-principal-component-analysis

[10] Slides of course Artificial Intelligence: Machine Learning and Pattern Recognition taken by Prof. Marcello Pelillo.

[11] Normalized cut: https://en.wikipedia.org/wiki/Segmentation-based_object_categorization#Normalized_cuts

[12] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and algorithm. NIPS 14 (2002).