Ca’ Foscari
University
Venezia

[CM0623-2] FOUNDATIONS OF ARTIFICIAL INTELLIGENCE (CM90)

Clustering

Student
Serena Zanon
Matricola 887050

Academic Year
2023 / 2024

Indice

Introduction

1.1 The assignment

The proposed assignment asks to execute clustering classification over the MNIST database.
Following, here is the assignment request:
Perform classification of the MNIST database (or a sufficiently small subset of it) using:

- Mixture of Gaussians with diagonal covariance (Gaussian Naive Bayes with latent class label);
- Mean shift;
- Normalized cut.

The unsupervised classification must be performed at varying levels of dimensionality reduction through PCA (say going from 2 to 200), in order to asses the effect of the dimensionality in accuracy and learning time.
Provide the code and the extracted clusters as the number of clusters k varies from 5 to 15, for the mixture of Gaussians 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:

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

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.

Tip: the means of the Gaussian models can be visualized as a greyscale images after PCA reconstruction to inspect the learned model.

1.2 Project structure and the MNIST dataset

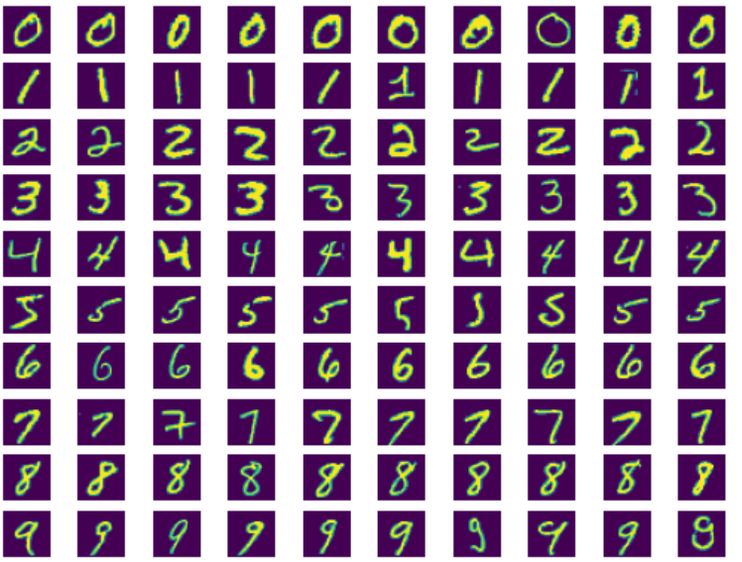
1.2.1 Project structure

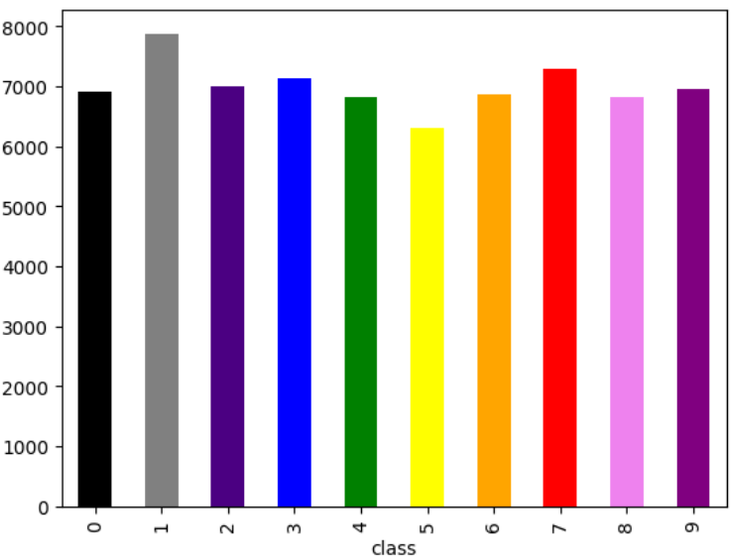
The project is given by several files and folders:

- The *latex* folder contains the latex transcriptions of the tables resulting from the executions, divided based on the adopted algorithm;
- The *results* folder contains three json files, one for each tested approach. Each json represents a list of dictionaries containing the necessary information to compute further analysis;
- *data.ipynb* is the introductive notebook file of the project in which the dataset is inspected;
- *GaussianMixture.py*, *GaussianMixture.ipynb* and *GaussianGreyScale.ipynb* are files related to the execution of the Gaussian Mixture approach;
- *MeanShift.py* and *MeanShift.ipynb* are files related to the execution of the Mean Shift approach;
- *NormalisedCut.py* and *NormalisedCut.ipynb* are files related to the execution of the Normalised Cut approach.

1.2.2 The MNIST dataset

The proposed dataset is the MNIST dataset and it is composed of 70.000 instances: a single instance is a 28x28 pixel image, so the number of the dataset's columns is 784.
In *data.ipynb*, the dataset is shrunk to a quarter of its original size due to high computational time and cost in performance. Here some elements composing the dataset and the distribution of the ten possible labels:




8000
7000
6000
5000
4000
3000
2000
1000
0
0
1
2
3
4
5
6
7
8
9
class

Unsupervised Learning

Unsupervised learning, also known as unsupervised machine learning, is a framework in machine learning in which algorithms learn patterns exclusively from unlabeled data: these algorithms discover hidden patterns or data groupings without the need for human intervention. [15]
Unsupervised learning has some issues, for instance an unsupervised learning task is considered more difficult than a supervised learning one, however this area of machine learning is still very useful: [19]

- Nowadays, large datasets are common and annotating them could be very costly and time consuming, so only a few examples are actually labelled;
- There may be cases in which the actual number of classes, into which the data is divided, is unknown;
- Some unsupervised learning approaches can be used to gain some extra information before designing a specific classifier.

There were algorithms designed specifically for unsupervised learning, such as clustering algorithms and dimensionality reduction techniques. [14]

2.1 Clustering

Clustering can be considered the most important unsupervised learning problem: it aims to form groups of homogeneous data points from a he-

terogeneous dataset, so it inspects the data structure to obtain relevant and significant information about the dataset's elements.[18]

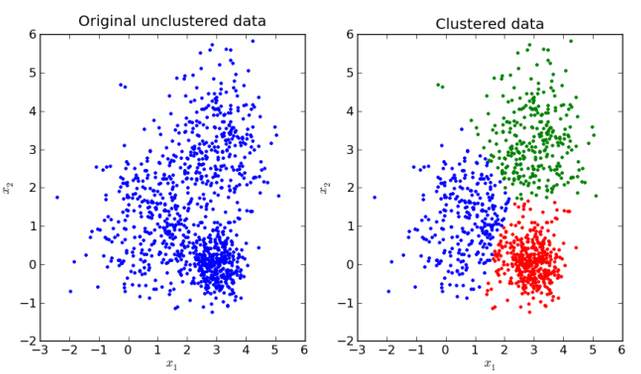
A simple way to clarify what is clustering could be with the following definition:

Clustering is the process of organising objects into groups whose members are similar in some way.

Indeed, it aspires to find clusters of elements: a cluster is therefore a collection of objects which are similar between them and are dissimilar to the objects belonging to other clusters.[15]

There are different types of clustering algorithms based on how they find clusters and on how they interpret the theoretical concept of cluster:[1]

- **Distribution-based clustering**: sometimes called probabilistic clustering, thihs approach groups together data points based on their probability distribution. For this type, Gaussian Mixture will be further analysed;
- **Density-based clustering**: it detects areas characterised by high concentrations of data objects bounded by areas of low concentrations of data objects (empty or sparse areas). For this type, Mean Shift will be further analysed;
- **Graph-based clustering**: it is based on the idea that data objects can be represented as graph's nodes, which can be divided into cohesive groups (clusters) based on their common characteristics. For this type, Normalised Cut will be further analysed.



2.1.1 Rand Index

The Rand index (or Rand measure) is a measure of the similarity between two data clusterings: it determines how well a cluster is built by comparing how pairs of data points are grouped together in the predicted cluster versus the true cluster.[12]

Given a set of n elements $S = \{o_1, \cdots, o_n\}$ and two partitions of S to compare:

- $X = \{X_1, \cdots, X_r\}$, a partition of S into *r* subsets,
- $Y = \{Y_1, \cdots, Y_s\}$, a partition of S into *s* subsets,

define the following: [11]

- *a*, the number of pairs of elements in S that are in the same subset in X and in the same subset in Y;
- *b*, the number of pairs of elements in S that are in different subsets in X and in different subsets in Y;
- *c*, the number of pairs of elements in S that are in the same subset in X and in different subsets in Y;
- *d*, the number of pairs of elements in S that are in different subsets in X and in the same subset in Y.

The Rand index, R, is:

$$R = \frac{a+b}{a+b+c+d} = \frac{a+b}{\binom{n}{2}}$$

Intuitively, $a+b$ can be considered as the number of agreements between X and Y and $c+d$ as the number of disagreements between X and Y.

The Rand Index varies between 0 and 1, where:

- A value of 1 indicates complete agreement between the two clusters, meaning all pairs of data points are either grouped together or apart in both clusterings.

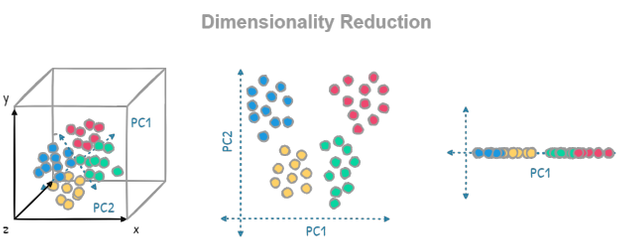
- A value of 0 suggests there's no agreement beyond what could be attributed to random chance.

Since the denominator is the total number of pairs, the Rand index represents the frequency of occurrence of agreements over the total pairs, or the probability that X and Y will agree on a randomly chosen pair. [11]

2.2 Dimensionality Reduction

Dimensionality reduction (or dimension reduction) techniques enhance machine learning models: it is a method for representing a given dataset using a lower number of features (i.e. dimensions) while still capturing the original data's meaningful properties.[2]

This is useful since it removes irrelevant or redundant features, or simply noisy data, to create a model with a lower number of variables which leads to higher computational performance. Their goal is to transform high-dimensional spaces into low-dimensional space through variable extraction or combination.[3]



2.2.1 Curse of dimensionality

In machine learning, high-dimensional data refers to data with a large number of features or variables. The curse of dimensionality is a common problem in machine learning, where the performance of the model deteriorates as the number of features increases: there is an inverse relationship between increasing model dimensions and decreasing generalizability.

This is because as the number of model input variables increases, the model's space increases and the data becomes sparse making it harder to find an effective model able to correctly predict the intrinsic data patterns. In addition, high-dimensional data can also lead to overfitting: the

model fits the training data too closely and does not generalise well to new data.

Collecting more data can reduce data sparsity and thereby offset the curse of dimensionality, however this is not always feasible.

Thus, the need for dimensionality reduction.[2]

2.2.2 PCA

PCA is the acronym for Principal Component Analysis and it is a linear dimensionality reduction technique: the data is linearly transformed onto a new coordinate system such that the directions capturing the largest variation in the data can be easily identified. [9]

It does this by transforming potentially correlated variables into a smaller set of variables, called principal components.

Principal components are a few linear combinations of the original variables that maximally explain the variance of all the variables. In the process, the method provides an approximation of the original data table using only these few major components.[17]

More specifically, the principal components of a collection of points in a real coordinate space are a sequence of p unit vectors, where the i-th vector is the direction of a line that best fits the data while being orthogonal to the first (i - 1) vectors.

Following the mathematical way to find principal components: [25]

Let the data be a set of D-dimensional vectors $\{x_1, \cdots, x_n\}$. The first principal component is a unit vector μ (i.e., $\mu^T\mu$ = 1) that maximises the variance of the projected data:

$$\mu^T S\mu = \frac{1}{n}\sum_{i=1}^{n}(\mu^T x_i - \mu^T\overline{x})^2$$

Where

- S is the covariance matrix

$$S = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})(x_i - \overline{x})^T$$

- $\overline{x}$ is the mean of the data, such that:

$$S = \frac{1}{n} \sum_{i=1}^{n} x_i$$

The unit vector μ represents the leading eigenvector of the matrix S and it defines the direction in which the data exhibits the most variance.
After identifying the first principal component, it's possible to find additional components by looking for new directions that maximise the variance among the vectors orthogonal to the directions already considered. Together, these eigenvectors form the basis for principal component analysis (PCA) while reducing the dimensionality of the data and preserving as much variance as possible.
PCA also minimises, or altogether eliminates, common issues such as multicollinearity and overfitting:[10]

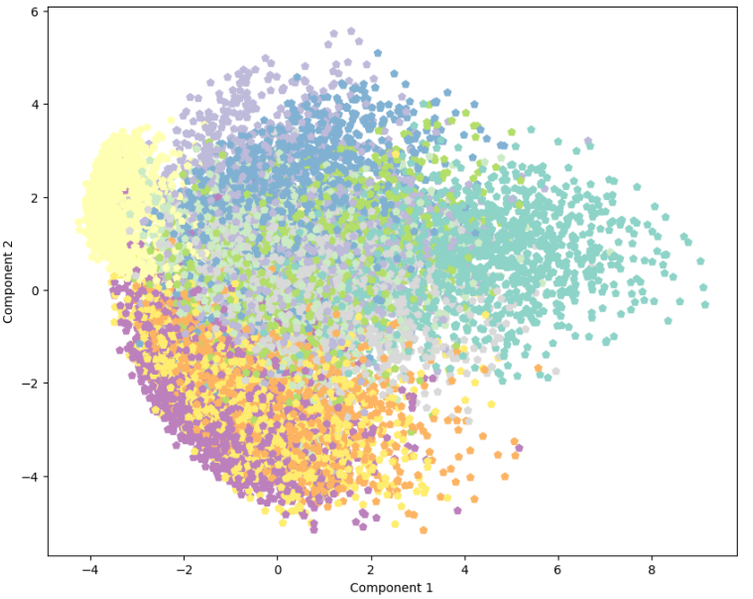
- Multicollinearity occurs when two or more independent variables are highly correlated with one another, which can be problematic for causal modelling;

- Overfit models will generalise poorly to new data, diminishing their predictive performance.

2.2.3 *data.ipynb*

In *data.ipynb*, there is the first application of PCA of the whole project and, specifically, it is used to visualise the shrunk dataset in two dimensions to get an idea of what it looks like.
Following the code related to the PCA's application and the resulting plot.

```
# Computing Pricipal Component Analysis
pca = PCA(n_components = 2)
print("Computing PCA with ", 2," components ...")
X_pca = pca.fit_transform(X)
```



The most important parameter of the *PCA* method is *n_components* which helps in specifying the number of principal components on which algorithms can rely on.

In this project, the *n_components* parameter assumes these values:

- For the Gaussian Mixture model, it assumes values between 2 and 200 with jumps of 5;
- For the Mean Shift and the Normalised Cut, it assumes values between 2 and 200 with jumps of 10 due to the high computational cost of the algorithms.

Gaussian Mixture

3.1 Gaussian Mixture Model

The Gaussian Mixture Model (GMM) is a probabilistic model used for clustering and density estimation. It assumes that the data is generated from a mixture of several Gaussian components, each representing a distinct cluster. [20]

It is a soft clustering approach meaning that each data point can belong to more than one cluster simultaneously.

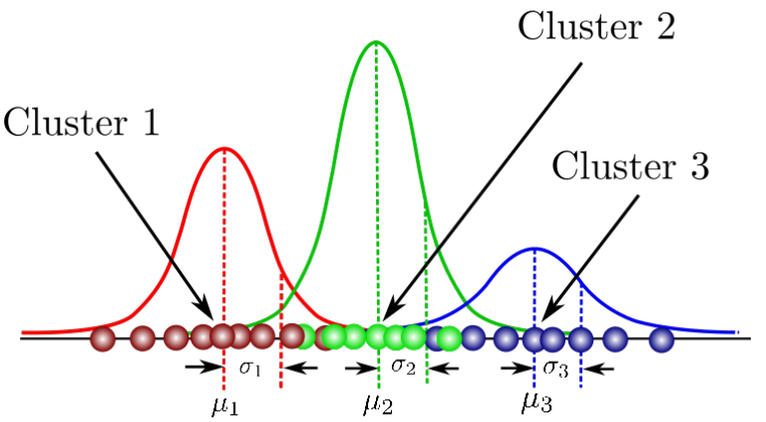
The Gaussian Mixture Model can be defined as a function composed of several Gaussians, each identified by $k \in \{1, \cdots, K\}$, where K is the number of clusters of our dataset.[5]

Each Gaussian k in the mixture is given by the following parameters:[5]

- A mean μ_k defining the center around which the data points are most densely clustered;
- A covariance $\sum_k$ defining its shape and orientation;
- A mixing probability π_k (or mixing coefficients) defining how big or small the Gaussian function will be or, in other words, the probability that a given observation is explained by Gaussian k.

Since the mixing coefficients are themselves probabilities, they must meet this condition:

$$\sum_{i=1}^{K} \pi_i = 1$$



Let's explain how the Gaussian Mixture Model actually works.[5] [24]

Let's define the probability that a data point *x* comes from Gaussian k as:

$$\mathbb{P}(z_k = 1|x)$$

which can be read as the probability that z_k is equal to 1 given the data point *x*. The variable z_k is a binary variable, such that:

$$z_k = \begin{cases} 1, \text{ when } x \text{ comes from Gaussian } k \\ 0, \text{ otherwise} \end{cases}$$

Likewise, the following stands:

$$\pi_k = \mathbb{P}(z_k = 1),$$

which means that the probability of observing a point that comes from Gaussian k is equivalent to the mixing coefficient for that Gaussian; so, the bigger the Gaussian is, the higher the probability will be.

Now, let's define Z as the set of all possible latent variables z_k:

$$Z = \{z_1, \cdots, z_K\}$$

Let's point out that each z_k occurs independently of others and they assume value 1 only when k is equal to the cluster the point comes from, thus the following is true:

$$\mathbb{P}(Z) = \mathbb{P}(z_1 = 1)^{z_1}\mathbb{P}(z_2 = 1)^{z_2} \cdots \mathbb{P}(z_K = 1)^{z_K} = \prod_{k=1}^{K} \pi_k^{z_k}$$

This means that finding the probability of observing the given data from Gaussian k is equal to the Gaussian function itself and, with the same logic, the following can be written:

$$\mathbb{P}(x|Z) = \prod_{k=1}^{K} \mathcal{N}(x|(\mu_k, |\sum_k))^{z_k}$$

where $\mathcal{N}$ represents the Gaussian distribution.

Finally, the equation that defines a Mixture of K Gaussians for each observation can be derived:

$$\mathbb{P}(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|(\mu_k, \sum_k))$$

Trivial to see that it depends on all the previously mentioned parameters $(\mu_k, \sum_k, \pi_k)$.[5]

3.2 Expectation - maximisation (EM) algorithm

Since the Gaussian Mixture Model requires, for each Gaussian, the estimation of the mean and the standard deviation parameters, one of the most common techniques for estimating these parameters is the Expectation-maximisation algorithm.

This approach is an iterative method to find maximum likelihood of parameters in statistical models depending on unobserved latent variables and it is characterised by two main steps:[16]

- **Expectation Step** (E-Step): Estimation of the latent variables based on the current parameters of the model;
- **Maximisation Step** (M-Step): Update of the model parameters to maximise the likelihood, thanks to the estimates of the E-step.

This process is repeated until convergence, point where the parameters no longer change significantly between iterations.

Let's dive deeper into the application of the EM method for the Gaussian Mixture Model.

First of all, let's define:

- The dataset $X = \{x_1, \cdots, x_N\}$;
- The number of clusters K;
- The parameters to estimate $\theta = \{\pi_k, \mu_k, \sum_k\}$

The expectation-maximisation iterative algorithm follows these steps:

1. **Initialization Step**: Start with initial guesses for the parameters $\theta^{(0)}$; these guesses can be done through different kinds of approaches;
2. **Expectation Step** (E-Step): The responsibilities γ_{ik} are computed and they represent the probability that data point x_i belongs to the k-th Gaussian component. The formula for γ_{ik} is as follows:

$$\gamma_{ik} = \frac{\pi_k \mathcal{N}(x_i|(\mu_k, \sum_k))}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i|(\mu_j, \sum_j))}$$

3. **Maximization Step** (M-Step): The parameters $\theta = \{\pi_k, \mu_k, \sum_k\}$ are updated to maximise the log-likelihood of the data, given the current responsibilities γ_{ik}. The updates are done as follows:
   - Updating the mixing coefficients:

$$\pi_k = \frac{1}{N} \sum_{i=1}^{N} \gamma_{ik}$$

   - Updating the means:

$$\mu_k = \frac{\sum_{i=1}^{N} \gamma_{ik} x_i}{\sum_{i=1}^{N} \gamma_{ik}}$$

   - Updating the covariance matrices:

$$\sum_k = \frac{\sum_{i=1}^{N} \gamma_{ik} (x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^{N} \gamma_{ik}}$$

4. **Repeat**: Repeat expectation and maximisation steps until convergence.

Here the formula of log-likelihood to maximise:

$$log\mathcal{L}(\theta|X) = \sum_{i=1}^{N} log(\sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|(\mu_k, \sum_k)))$$

EM stops when the log-likelihood no longer improves significantly between iterations.[5]

3.3 Implementation and results

In the presented project, there are three files dedicated to the Gaussian Mixture Model:

- *GaussianMixture.py* in which is present the actual implementation of the model;
- *GaussianMixture.ipynb* in which some analysis over the executions' results are made;
- *GaussianGreyScale.ipynb* in which a further analysis over the means of the Gaussians is made.

3.3.1 *GaussianMixture.py*

In this python file, there is only one function called *Gaussian_model*:

- It takes in input the dataset and its labels;
- It returns a list of dictionaries: each of them refers to a specific execution of the Gaussian Mixture Model and contains these information: the number of adopted PCA, the number of clusters, the value of the rand index, the fit and the predict time.

The core of this function is made by few rows:

- **PCA computation**

```
pca = PCA(n_components = n_features)
X_pca = pca.fit_transform(X)
```

- **Guassian Mixture Model computation**

```
model = GaussianMixture(n_components = n_clusters,
covariance_type = "diag", random_state = 42,
max_iter = 300)
```

```
model.fit(X_pca)

predictions = model.predict(X_pca)
```

Here an in-depth explanation of the *GaussianMixture* method:

- **n_components**: the default value is '1' and it represents the number of mixture components (clusters);
- **covariance_type**: the default value is 'full' and it is a string describing the type of covariance parameters to use. The chosen one is 'diag': each component has its own diagonal covariance matrix. Other possible values are: 'full', 'tied', 'spherical';
- **random_state**: the default value is 'None' and it controls the random seed given to the method chosen to initialize the parameters;
- **max_iter**: the default value is '100' and it indicates the number of EM iterations to perform.

This model is run over the MNIST dataset: the number of principal components varies between 2 and 200 with jumps of 5 and the *n_components* parameter takes values in range (5, 15), including extremes.

3.3.2 *GaussianMixture.ipynb*

In this python notebook, the analysis over the Gaussian Mixture Model executions are made.
First of all, the best and the worst settings are retrieved based on the rand index.

BEST executions

PCA	N clusters	Rand index	Fit time	Predict time
2	15	0.872870	0.127355	0.000000

PCA	N clusters	Rand index	Fit time	Predict time
7	14	0.887346	0.184345	0.000000
12	14	0.893416	0.172364	0.005907
17	15	0.895604	0.371804	0.000000
22	15	0.901297	0.469498	0.000000
27	15	0.899027	0.484370	0.007212
32	15	0.899339	0.356417	0.000000
37	15	0.880776	0.650051	0.007931
42	15	0.880784	0.800008	0.000000
47	15	0.878379	1.140891	0.000000
52	15	0.880761	0.828157	0.000000
57	15	0.866724	0.851296	0.000000
62	15	0.862163	0.836393	0.012506
67	13	0.855251	0.875170	0.016524
72	14	0.865340	0.888834	0.017455
77	15	0.858273	0.917454	0.021590
82	15	0.846651	1.981160	0.011947
87	15	0.856814	1.296520	0.015623
92	15	0.860312	1.060688	0.004992
97	15	0.846079	1.072114	0.015685
102	15	0.845612	1.977097	0.015604
107	15	0.859640	1.381130	0.017587
112	15	0.837566	1.945226	0.013707
117	14	0.817007	2.171170	0.017532
122	15	0.858821	1.531610	0.025364
127	15	0.856541	1.284711	0.028878
132	15	0.849770	1.855718	0.026613
137	13	0.833688	1.330014	0.023069
142	14	0.840526	2.108474	0.032185

PCA	N clusters	Rand index	Fit time	Predict time
147	10	0.813139	1.023107	0.015695
152	14	0.817004	2.151903	0.025705
157	15	0.824006	1.798108	0.019230
162	15	0.814412	2.476744	0.031765
167	13	0.818070	1.438401	0.025473
172	11	0.829348	1.628730	0.029451
177	15	0.820542	2.879212	0.023937
182	15	0.843565	2.096602	0.025191
187	14	0.842686	2.590132	0.023979
192	12	0.822782	3.444998	0.031559
197	14	0.830897	2.053268	0.026960

The best settings are:

- **PCA**: 22
- **n_clusters**: 15
- **rand_index**: 0.90129
- **fit_time**: 0.469498
- **predict_time**: 0.000000

WORST executions

PCA	N clusters	Rand index	Fit time	Predict time
2	5	0.731715	0.216463	0.000000
7	5	0.789100	0.087681	0.001457
12	5	0.766221	0.122249	0.000000
17	7	0.663835	0.461569	0.000000
22	5	0.751724	0.175101	0.000000

PCA	N clusters	Rand index	Fit time	Predict time
27	5	0.718477	0.326898	0.002990
32	5	0.727399	0.316226	0.000000
37	5	0.719792	0.412350	0.000000
42	7	0.702501	0.350211	0.000000
47	7	0.698408	0.395598	0.015691
52	7	0.716947	0.499527	0.016035
57	5	0.724108	0.402075	0.000000
62	7	0.757244	0.477875	0.000000
67	7	0.757141	0.515217	0.009129
72	7	0.755359	0.562124	0.015622
77	5	0.758177	0.372230	0.000000
82	6	0.755626	0.645031	0.018867
87	5	0.722178	0.423302	0.015040
92	7	0.720676	1.335056	0.000000
97	5	0.723348	0.501524	0.015623
102	5	0.750081	0.492269	0.015764
107	7	0.714329	1.564752	0.000000
112	5	0.712281	0.984777	0.000000
117	5	0.711517	0.935383	0.012510
122	5	0.725297	0.649659	0.024819
127	5	0.723218	0.655376	0.014354
132	5	0.723812	0.647899	0.012390
137	5	0.723305	0.704585	0.023337
142	5	0.706706	1.043385	0.022505
147	5	0.710750	0.959540	0.028154
152	5	0.713035	0.909401	0.018763
157	5	0.712664	0.951772	0.019604
162	5	0.711139	1.066298	0.017690

PCA	N clusters	Rand index	Fit time	Predict time
167	5	0.708823	1.108466	0.021361
172	5	0.730122	1.358721	0.025798
177	5	0.717440	0.969714	0.017416
182	5	0.717600	0.921145	0.023660
187	5	0.729989	1.818280	0.019099
192	6	0.738840	1.012340	0.020422
197	5	0.717455	1.091049	0.019976

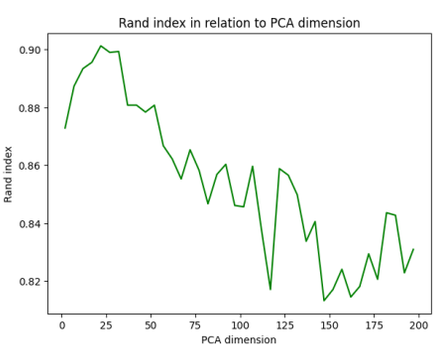
The worst settings are:

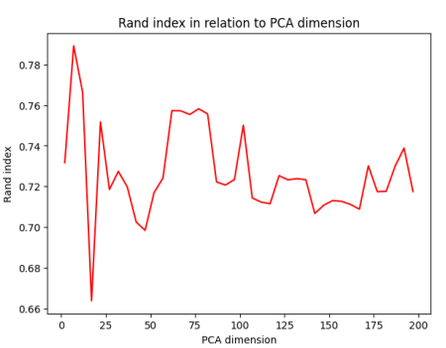
- **PCA**: 17
- **n_clusters**: 7
- **rand_index**: 0.663835
- **fit_time**: 0.461569
- **predict_time**: 0.000000

Analysis results

In this paragraph, plots of the best (green) and the worst (red) executions are shown in order to better understand hidden behaviours of the model over the data.

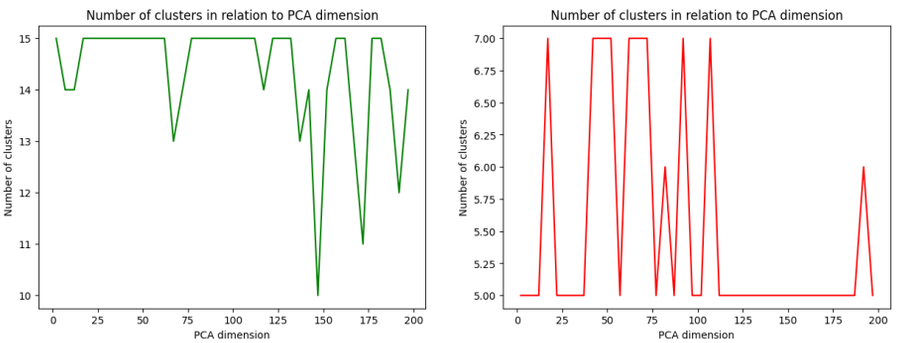
- **Rand index in relation to PCA dimension**





It can be seen that the rand index, which can be considered the score metric of these executions, decreases with increasing PCA values; plus, this behaviour can be noted even in the worst cases. This trend could be explained by the fact that increasing the number of PCA may lead to the inclusion of components that contain noise rather than useful information.

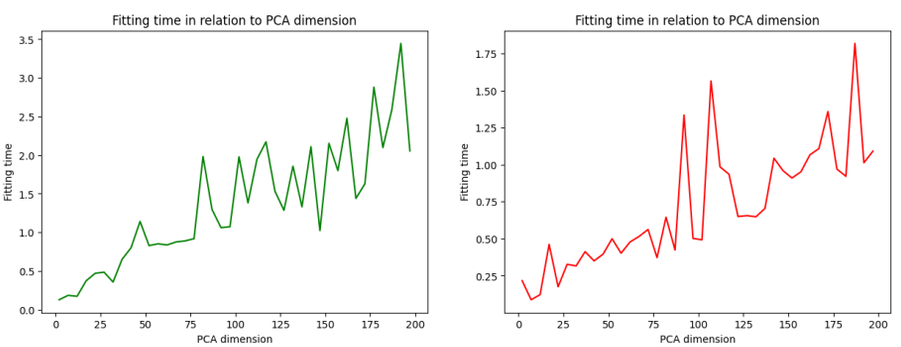
- **Number of clusters in relation to PCA dimension**



From these graphs, it's possible to understand the following:

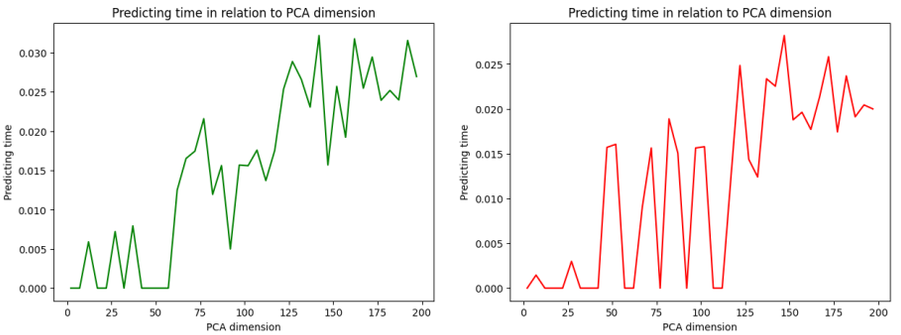
- Most of the times, the Gaussian Mixture Model performs well with higher values of clusters (*n_components* parameter);
- The worst executions are characterised by lower number of clusters.

- **Fitting time in relation to PCA dimension**



The two trends are quite similar meaning that the fitting time, so the time required by the training phase, gets higher when the number of adopted PCA is higher. This could be the consequence of the fact that more input dimensions implies more computational time.

- **Predicting time in relation to PCA dimension**



Although the trends are less clear than those for fitting time, it can be guessed that the time taken to produce the predictions increases as the number of principal components increases.
However, these not-so-linear trends can be explained: they are characterised by a large variance and this is caused by the EM algorithm itself since it is sensitive to the initialization step.

GaussianGreyScale.ipynb

As already mentioned in the first chapter of this report, there is a third file about the Gaussian Mixture Model completely dedicated to obtain grey scale images from the resulting means of the executions.
In order to do so, the *Gaussian_model* method is replicated with the exact same conditions over the entire dataset:

- PCA goes from 2 to 200 with jumps of 5;
- The *n_components* parameter takes values in range (5, 15).

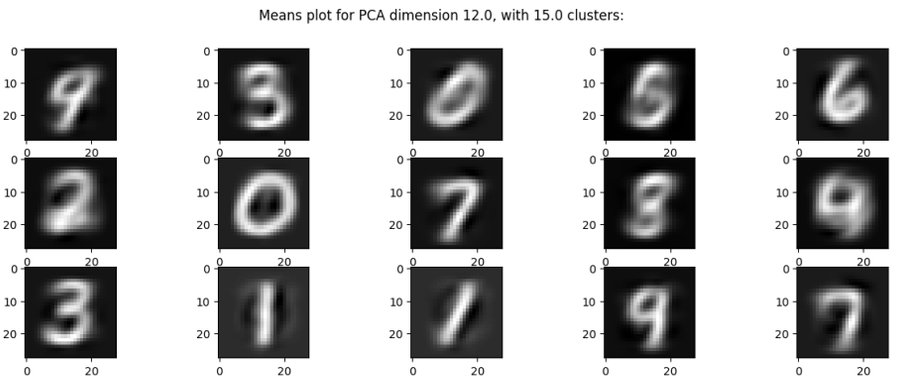
As a result of this computation, the best settings are:

- **PCA**: 12
- **n_clusters**: 15
- **rand_index**: 0.897154

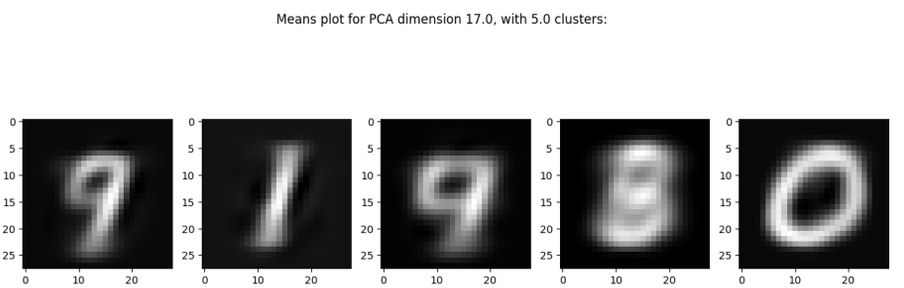
and the worst settings are:

- **PCA**: 17
- **n_clusters**: 5
- **rand_index**: 0.66814

Grey scaled images obtained with the best settings



Grey scaled images obtained with the worst settings



The second images are far less precise than the first ones; indeed, in the grey scale images retrieved thanks to the best settings, digits can be distinguished in an easier way with respect to the ones retrieved from the worst settings.

Mean Shift

4.1 Mean Shift

Mean shift is a non-parametric feature-space mathematical analysis technique: it moves data points towards regions of maximum density, using a search window and a kernel.
It iteratively assigns the data points to clusters by shifting points towards the mode (mode is the highest density of data points in the region, in the context of the Mean Shift). As such, it is also known as the Mode-seeking algorithm.[8]
Mean Shift behaves as follows:[8]

- Given a set of data points, the algorithm iteratively assigns each data point towards the closest cluster centroid and the direction to the closest cluster centroid is determined by where most of the points nearby are at.

- In each iteration, each data point will move closer to where the most points are at, which will lead to the cluster center. When the algorithm stops, each point is assigned to a cluster.

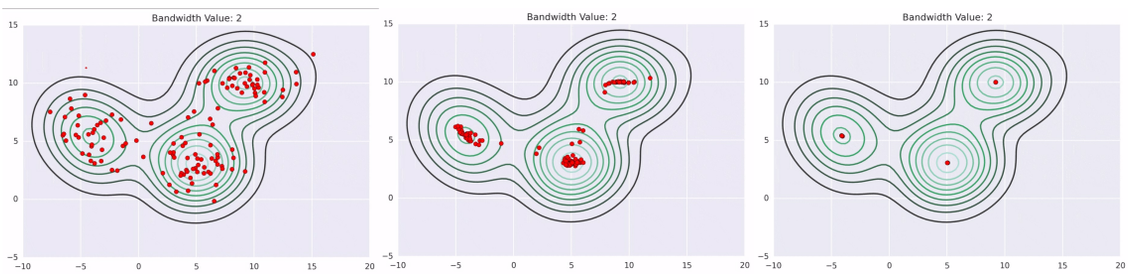
Let's define the algorithm's steps:[23]

1. **Initialization Step**: Create a cluster for each single data point of the dataset;

2. **MeanShift Step**: Each cluster is shifted towards higher density regions by shifting their centroid to the data-points' mean within the cluster. This step will be repeated until no shift yields a higher density;

3. **Selection Step**: Select final clusters by deleting overlapping windows. When multiple clusters overlap, the cluster containing the most points is preserved, and the others are deleted.

4. **Assignment Step**: Assign the data points to the cluster in which they reside.



One of the main advantages of Mean Shift clustering is that it does not require the number of clusters to be specified beforehand. It also does not make any assumptions about the distribution of the data and can handle arbitrary shapes and sizes of clusters.
However, it can be sensitive to the choice of kernel and the radius of the kernel.[8]

4.2 Kernel density estimation

The term kernel is used in statistical analysis to refer to a window function; in particular, for Mean Shift, the kernel specifies the shape of the density distribution used to evaluate the proximity of points.
In order to estimate the probability density function of a random variable, Mean Shift exploits kernel density estimation (KDE).
Formally speaking,

Let $(x_1, x_2, \cdots, x_n)$ be independent and identically distributed samples drawn from some univariate distribution with an unknown density f at any given point x.

The focus is on estimating the shape of this function f.

Its kernel density estimator is:

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{x - x_i}{h})$$

where

- K is the kernel function;
- h $>$ 0 is the smoothing parameter called bandwidth or simply width.

There are many kernel functions, however the two most frequently used kernel profiles for Mean Shift are:

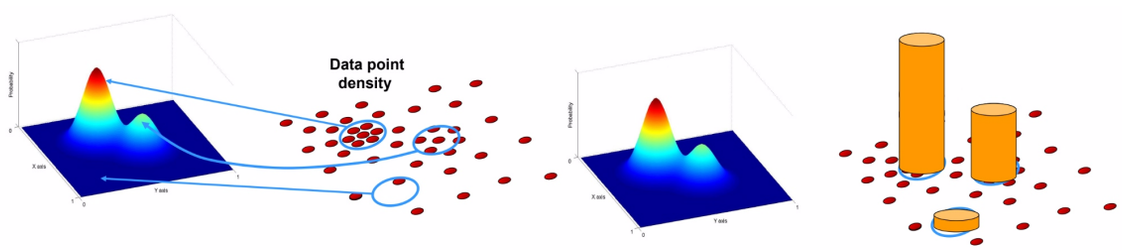
- **Flat Kernel**

$$\text{k(x)} = \begin{cases} 1, \text{ if } x \leq \lambda \\ 0, \text{ if } x > \lambda \end{cases}$$

- **Gaussian Kernel**

$$k(x) = e^{-\frac{x}{2\sigma^2}}$$

where the standard deviation parameter σ works as the bandwidth parameter, h.[7]



4.3 Implementation and results

In the presented project, there are two files dedicated to the Mean Shift Model:

- *MeanShift.py* in which is present the actual implementation of the model;
- *MeanShift.ipynb* in which some analysis over the executions' results are made;

4.3.1 *MeanShift.py*

In this python file, there is only one function called *MeanShift_model*:

- It takes in input the dataset and its labels;
- It returns a list of dictionaries: each of them refers to a specific execution of the Mean Shift Model and contains these information: the number of adopted PCA, the width of the kernel, the number of found clusters, the value of the rand index, the fit and the predict time.

The core of this function is made by few rows:

- **PCA computation**

```
pca = PCA(n_components = n_features)
X_pca = pca.fit_transform(X)
```

- **Mean Shift Model computation**

```
model = MeanShift(bandwidth = width, n_jobs = -1)
model.fit(X_pca)
predictions = model.predict(X_pca)
```

Here an in-depth explanation of the *MeanShift* method:

- **bandwidth**: the default value is 'None' and it represents the bandwidth used in the flat kernel;
- **n_jobs**: the default value is 'None' and it indicates the number of jobs to use for the computation. '-1' means using all processors.

This model is run over the MNIST dataset: the number of principal components varies between 2 and 200 with jumps of 10 and the *bandwidth* parameter takes the following values [0.2, 0.4, 0.6, 0.8, 1, 2, 5, 10, 15, 20].

4.3.2 *MeanShift.ipynb*

In this python notebook, the analysis over the Mean Shift Model executions are made.

First of all, the best and the worst settings are retrieved based on the rand index.

BEST executions

PCA	Width	N clusters	Rand index	Fit time	Predict time
2	0.200000	752	0.899545	20.726465	0.111947
12	2.000000	4102	0.903594	80.168944	0.662323
22	2.000000	10333	0.901745	14.331039	1.578656
32	2.000000	11832	0.901128	13.471279	1.696847
42	2.000000	12213	0.900817	15.529615	1.830008
52	2.000000	12404	0.900486	15.267745	1.826886
62	5.000000	1077	0.903654	103.697971	0.175156
72	5.000000	1454	0.906010	105.771794	0.247195
82	5.000000	1798	0.906893	111.291040	0.302584
92	5.000000	2109	0.906865	131.851504	0.335757
102	5.000000	2392	0.907016	128.294530	0.335901

PCA	Width	N clusters	Rand index	Fit time	Predict time
112	5.000000	2656	0.907373	122.302016	0.413057
122	5.000000	2909	0.907424	134.447090	0.443408
132	5.000000	3133	0.907781	168.028476	0.485799
142	5.000000	3367	0.907541	159.201806	0.515761
152	5.000000	3524	0.908042	175.233334	0.560470
162	5.000000	3711	0.907794	167.821794	0.597847
172	5.000000	3862	0.907593	193.510026	0.614365
182	5.000000	4025	0.907589	201.547781	0.664220
192	5.000000	4155	0.907903	225.981634	0.725093

The best settings are:

- **PCA**: 22
- **width**: 5
- **n_clusters**: 3524
- **rand_index**: 0.908042
- **fit_time**: 175.233334
- **predict_time**: 0.560470

WORST executions

PCA	Width	N clusters	Rand index	Fit time	Predict time
2	5	752	0.100227	12.279422	0.000000
12	5	4102	0.100227	52.700547	0.000000
22	10	10333	0.100227	30.154025	0.000000
32	10	11832	0.100227	42.857116	0.000000
42	10	12213	0.100227	56.068958	0.000000
52	10	12404	0.100227	69.777131	0.000000

PCA	Width	N clusters	Rand index	Fit time	Predict time
62	10	1077	0.100227	82.335043	0.000000
72	10	1454	0.100227	94.767410	0.000000
82	10	1798	0.100227	106.799918	0.000000
92	10	2109	0.100227	116.611817	0.015623
102	10	2392	0.100227	128.350755	0.000000
112	10	2656	0.100227	144.925427	0.015617
122	10	2909	0.100227	157.463581	0.000000
132	10	3133	0.100227	170.595663	0.015624
142	10	3367	0.100227	183.026297	0.000000
152	10	3524	0.100227	195.063022	0.000000
162	10	3711	0.100227	209.305946	0.000000
172	10	3862	0.100227	221.485626	0.000000
182	10	4025	0.100227	234.547239	0.015635
192	10	4155	0.100227	246.293927	0.015628

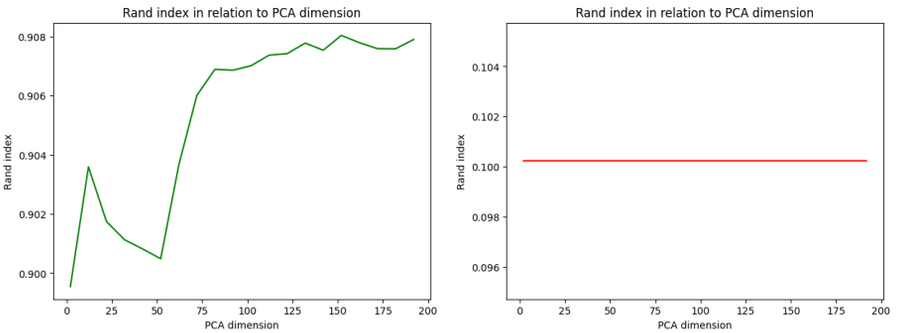
The worst settings are:

- **PCA**: 2
- **width**: 5
- **n_clusters**: 752
- **rand_index**: 0.100227
- **fit_time**: 12.279422
- **predict_time**: 0.000000

Analysis results

In this paragraph, plots of the best (green) and the worst (red) executions are shown in order to better understand hidden behaviours of the model over the data.

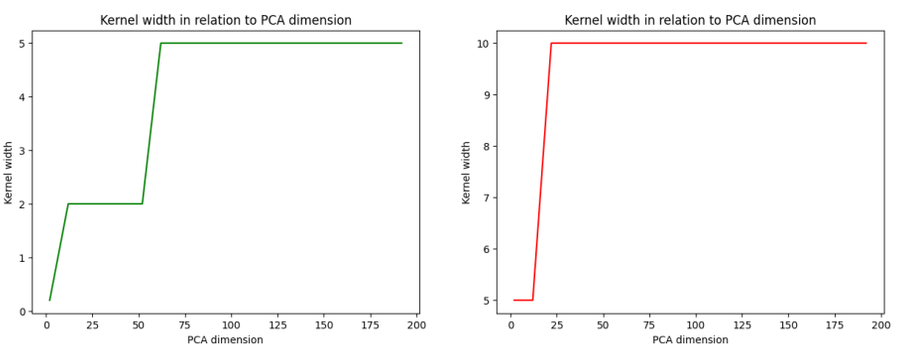
- **Rand index in relation to PCA dimension**



The trends of the rand index are quite different for the best and the worst settings:

- For the best settings, it's possible to capture an increasing trend, so as the number of PCA gets higher, the rand index gets higher. By increasing the number of main components, more information is included, preventing relevant features from being overlooked and a more accurate representation of density in the data can be understood, allowing the algorithm to better identify clusters.
- For the worst settings, the trend is constant equal to 0.100. This means that the score is extremely low suggesting that the model is not learning anything from the data.

- **Kernel width in relation to PCA dimension**

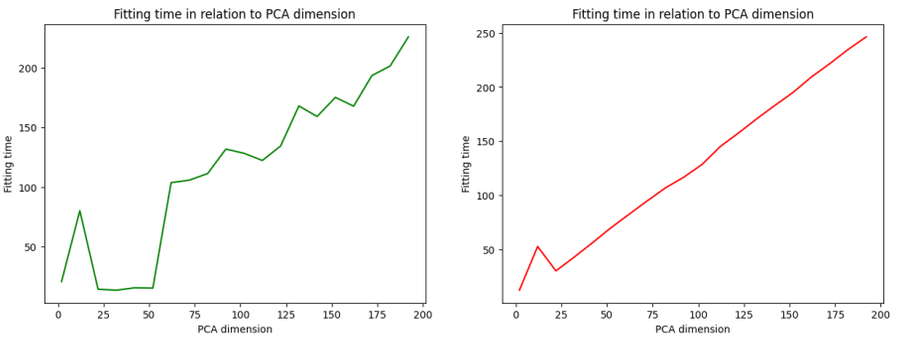


Thanks to these graphs, it's possible to obtain the information about the best and the worst value for the kernel width:

- Best value: 5

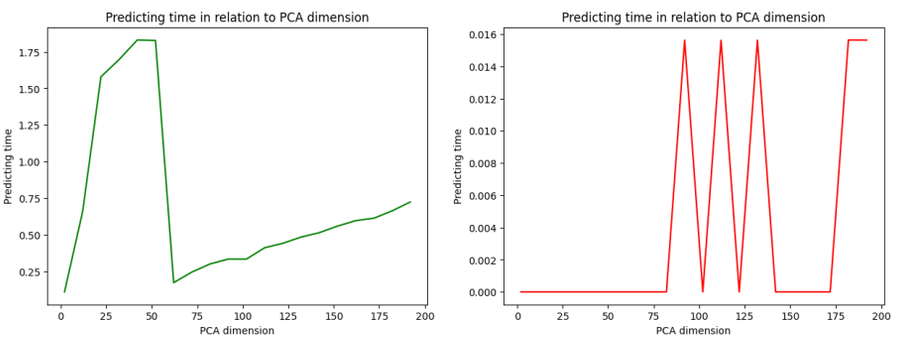
- Worst value: 20

- **Fitting time in relation to PCA dimension**



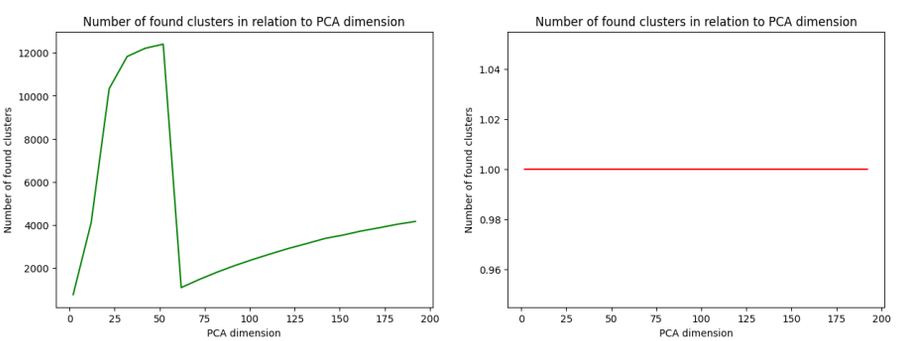
As for the previous model, increasing the number of PCA leads to higher computational time for fitting phase: more information need more time to be processed.

- **Predicting time in relation to PCA dimension**



Even if the trends are not linear, it's possible to catch the behind-trend of these graphs: as the number of PCA gets higher, the required prediction time gets higher.

- **Number of found clusters in relation to PCA dimension**



The information that can be retrieved from these graphs are:

- The number of found clusters tends to increase as the number of PCA increases; let's notice that this trend is extremely similar to the one of the best settings for predicting time, meaning that the number of clusters and the time needed for the predictions are strongly related;
- The worst executions are all characterised by 1 cluster.

Normalised Cut

Before explaining what is the Normalised Cut approach, an introduction about graph theory must be done.

5.1 Graph Theory

Graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects.
An undirected graph is made up of vertices (also called nodes or points) which are connected by edges (also called arcs, links or lines).
An undirected graph is given by an ordered pair $G = (V, E)$ comprising: [6]

- V, a set of vertices;
- $E \subseteq \{\{x, y\} | x, y \in V \wedge x \neq y\}$, a set of edges, which are unordered pairs of vertices.

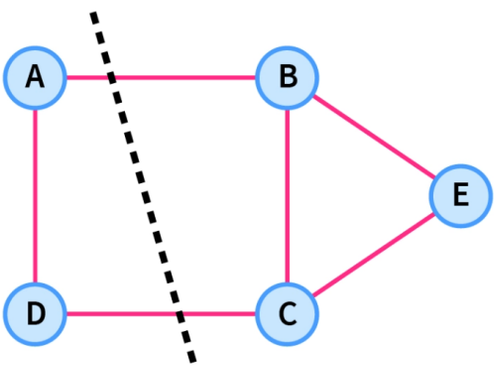
Furthermore, edges can be associated with a value which is interpreted as the weight of that specific edge. In this particular case, the graph is an undirected weighted graph. [6]
An undirected weighted graph is given by an ordered triple $G = (V, E, \phi)$ comprising: [6]

- V , a set of vertices;
- E, a set of edges;

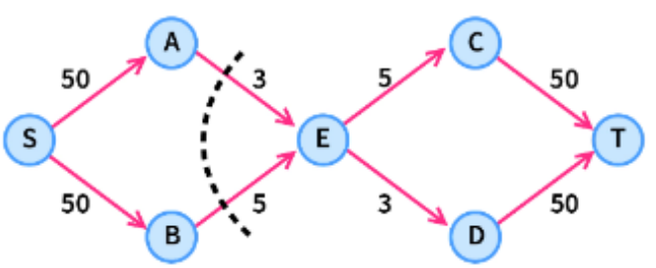
- $\phi : E \leftarrow \{\{x, y\}|x, y \in V \wedge x \neq y\}$, an incidence function mapping every edge to an unordered pair of vertices.

A cut $C = (A, B)$ is a partition of V of a graph $G = (V, E)$ into two subsets S and T. The cut-set of a cut $C = (A, B)$ is the set $\{(u, v) \in E|u \in A, v \in B\}$ of edges that have one endpoint in A and the other endpoint in B.
In an unweighted undirected graph, the size or weight of a cut is the number of edges crossing the cut. [18]

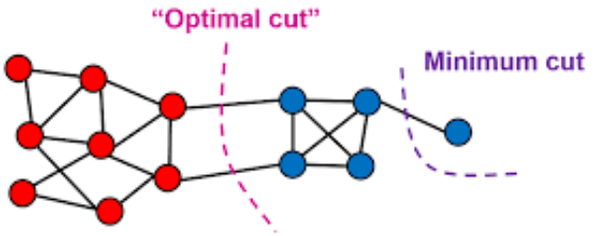
$$C(A, B) = \sum_{i \in A, j \in B} w(i, j)$$



The optimal bi-partitioning of a graph is the one that minimises this cut value.
Although there are an exponential number of such partitions, finding the minimum cut of a graph is a well studied problem, and there exist efficient algorithms for solving it.



However, the minimum cut criteria favours cutting small sets of isolated nodes in the graph.



5.2 Normalised Cut

To avoid this unnatural bias for partitioning out small sets of points, the normalised cut can be adopted:[22]

$$Ncut(A, B) = \frac{C(A,B)}{vol(A)} + \frac{C(A,B)}{vol(B)}$$

where:

- $vol(S) = \sum_{i \in S} d_i$, so the volume of the partition S;
- $d_i = \sum_j w(i, j)$, so the degree of node i.

Unfortunately, identifying the minimum normalised cut is NP-hard.

5.3 Graph Laplacian

There exists an efficient approximation for the minimum normalized cut problem using linear algebra. This approach is based on the Graph Laplacian (Laplacian Matrix).

The Laplacian matrix is defined as follows: [21]

Given a graph $G = (V, E)$ where V is the set of nodes and E is the set of edges, with an adjacency matrix $A \in \{0, 1\}^{nxn}$, where [4]

$$A_{i,j} = \begin{cases} \text{1,if there is an edge between } v_i \text{ and } v_j \\ \text{0, otherwise} \end{cases}$$

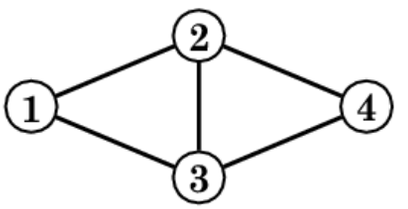
and degree matrix $D \in \mathbb{Z}^{nxn}$, where

$$D_{i,j} = \begin{cases} \text{degree}(v_i), \text{if i = j} \\ 0, \text{otherwise} \end{cases}$$

The Laplacian matrix is defined as:

$$L = D - A$$

$$D = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$



$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \qquad L = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix}$$

For all vectors f in $\mathbb{R}^n$, this equality must be satisfied:

$$f^T L f = \frac{1}{2} \sum_{i,j=1}^{n} w_{i,j}(f_i - f_j)^2$$

In addition, L has some interesting properties:

- L is a symmetric matrix (by assumption);
- $f^T L f \geq 0$;
- L is a positive semi-definite matrix and its smallest eigenvalue is 0 and the corresponding eigenvector is a vector composed by n 1s ($0 = \lambda_1 \leq \lambda_2 \leq \cdots$);
- The multiplicity of the smallest eigenvalue is equal to the number of connected components in the graph; this gives some information about the internal structure of the graph itself simply by looking at the eigenvalues

The problem is finding the best Normalised Cut and it can be formulated as follows: [21]
Given an indicator vector representing a cut:

$$x_i = \begin{cases} +1, i \in A \\ -1, i \in B \end{cases}$$

Where A and B are two partitions originating from the cut, find among all possible indicator vectors the one which minimises the Normalised Cut:

$$\min_x Ncut(x) = \min_y \frac{y'(D-A)y}{y'Dy}$$

with such constraints $y'D1 = \sum_i y_i d_i = 0$, where $y_i \in \{1, -b\}$ (b is a constant).

5.4 Normalised cut as an eigensystem

The problem above deals with discrete values so it is hard to solve; however, it's possible to relax it and transform it in a real problem going from this

$$\min_y \frac{y'(D-A)y}{y'Dy}$$

to this

$$\min_y y'(D-A) \text{ with } y'Dy = 1$$

Summarizing, the generalised eigenvalue problem is

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

There are many advantages while doing this procedure:

- First of all, it's easier to solve than the original problem;
- It gives a good approximation of the original discrete problem

Nevertheless, it's not sure that the final solution is the correct one. [21]

5.5 Spectral clustering

Spectral clustering techniques exploit the idea above in order to partition data.

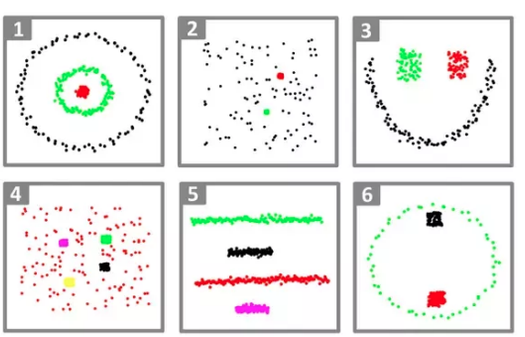
The basic steps of the algorithm are the following: [13]

1. Construct a similarity graph and calculate the Laplacian matrix L;
2. Calculate the k smallest eigenvectors $u_1, u_2, \cdots, u_k$ of L;
3. Let $U = [u_1 u_2 \cdots u_k] \in \mathbb{R}^{nxk}$;
4. Normalise the rows of U to norm 1:

$$U_{ij} \leftarrow \frac{U_{ij}}{(\sum_k U_{ik}^2)^{\frac{1}{2}}}$$

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 algorithms.

This algorithm performs well even with non-spherical shape clusters, however it is sensitive to outliers: since it creates partitions, every point must be assigned to a cluster.



5.5.1 Choice of *k*

An important pre-step of the algorithm is given by the choice of k's value; indeed, it should be chosen in such a way that the first k eigenvalues are small and close to each other while the (k+1)-th eigenvalue is larger,

meaning that the gap between λ_k and λ_{k+1} should be big. [21]
Trivially, two possible situations can be faced:

- If the clusters are well separated, it's easy and quite trivial to identify the best value for k;
- If the clusters are close to each other, it's more challenging to find the value of k: the eigenvalues could be equally spaced from each other; so, it's more difficult to determine which one returns the best performance

5.6 Implementation and results

In the presented project, there are two files dedicated to the Normalised Cut Model:

- *NormalisedCut.py* in which is present the actual implementation of the model;
- *NormalisedCut.ipynb* in which some analysis over the executions' results are made;

5.6.1 *NormalisedCut.py*

In this python file, there is only one function called *NormalisedCut_model*:

- It takes in input the dataset and its labels;
- It returns a list of dictionaries: each of them refers to a specific execution of the Normalised Cut Model and contains these information: the number of adopted PCA, the number of clusters, the value of the rand index, the fit-predict time.

The core of this function is made by few rows:

- **PCA computation**

```
pca = PCA(n_components = n_features)
X_pca = pca.fit_transform(X)
```

- **Normalised Cut Model computation**

```
model = SpectralClustering(n_clusters = n_clusters,
affinity = "nearest_neighbors",
n_neighbors = 30, n_jobs = -1)
predictions = model.fit_predict(X_pca)
```

Here an in-depth explanation of the *SpectralClustering* method:

- **n_clusters**: the default value is '8' and it indicates the dimension of the projection subspace.;
- **affinity**: the default value is 'rbf' and it explains how to construct the affinity matrix. The adopted value is 'nearest_neighbors' which performs the construction of the affinity matrix by computing a graph of nearest neighbors. Other possible values are: 'precomputed', 'precomputed_nearest_neighbors';
- **n_neighbors**: the default value is '10' and it represents the number of neighbors to use when constructing the affinity matrix using the nearest neighbors method;
- **n_jobs**: the default value is 'None' and it indicates the number of jobs to use for the computation. '-1' means using all processors.

This model is run over the MNIST dataset: the number of principal components varies between 2 and 200 with jumps of 10 and the *n_clusters* parameter takes values in range (5, 15), including extremes.

5.6.2 *NormalisedCut.ipynb*

In this python notebook, the analysis over the Normalised Cut Model executions are made.
First of all, the best and the worst settings are retrieved based on the rand index.

BEST executions

PCA	N clusters	Rand index	Fit-predict time
2	15	0.871916	9.968873
12	13	0.901773	93.925697
22	13	0.928241	97.104162

PCA	N clusters	Rand index	Fit-predict time
32	13	0.929651	93.535461
42	13	0.929304	118.644037
52	13	0.929534	84.306271
62	13	0.928363	89.661510
72	13	0.928123	140.997974
82	13	0.927270	132.329518
92	13	0.926467	92.206803
102	13	0.926266	121.380501
112	13	0.926366	141.488630
122	13	0.926062	123.511004
132	13	0.925674	125.605255
142	13	0.925744	115.471858
152	13	0.925330	134.835099
162	13	0.925188	199.984154
172	13	0.924943	126.554568
182	13	0.924455	113.783566
192	13	0.924557	130.116361

The best settings are:

- **PCA**: 32
- **n_clusters**: 13
- **rand_index**: 0.929651
- **fit-predict time**: 93.535461

WORST executions

PCA	N clusters	Rand index	Fit-predict time
2	5	0.775228	10.124946
12	5	0.721537	88.893354
22	5	0.733339	85.485062
32	5	0.734934	96.552469
42	5	0.735248	119.664068
52	5	0.780706	84.633018
62	5	0.792947	90.187418
72	5	0.796712	141.308990
82	5	0.797973	133.021512
92	5	0.796743	93.475973
102	5	0.799136	121.933798
112	5	0.800619	142.029368
122	5	0.800375	124.639915
132	5	0.800532	126.923462
142	5	0.799792	116.134729
152	5	0.800874	134.357249
162	5	0.799774	199.504622
172	5	0.798850	126.142573
182	5	0.800160	113.739285
192	5	0.797922	130.354172

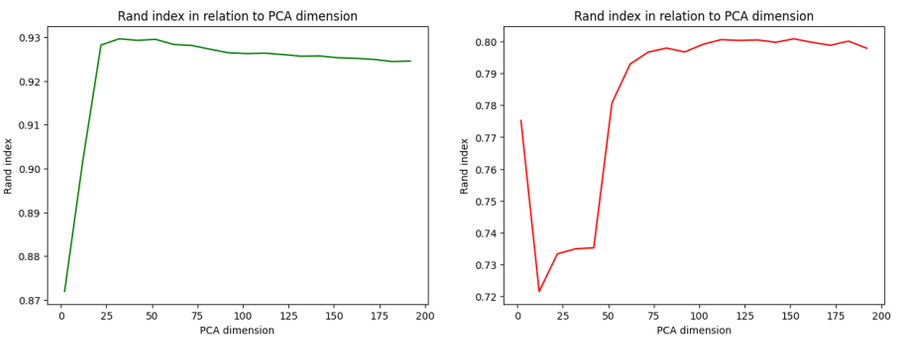
The worst settings are:

- **PCA**: 12
- **n_clusters**: 5
- **rand_index**: 0.721537
- **fit-predict_time**: 88.893354

Analysis results

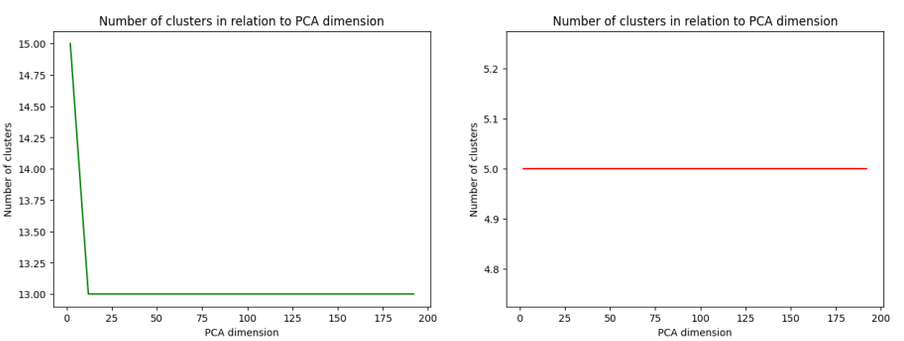
In this paragraph, plots of the best (green) and the worst (red) executions are shown in order to better understand hidden behaviours of the model over the data.

- **Rand index in relation to PCA dimension**



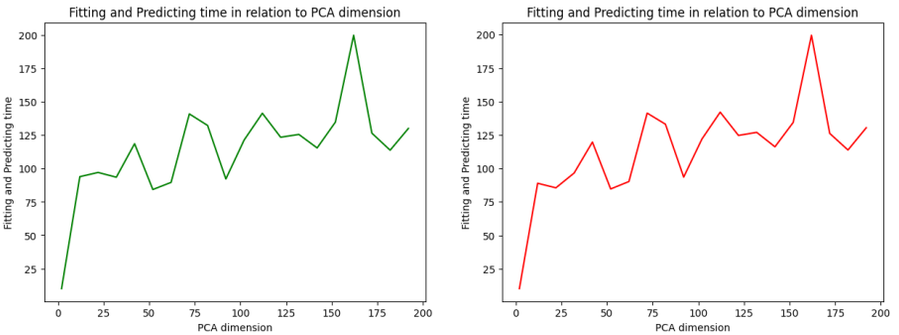
Conceptually, the two trends have same behaviour: with an increasing number of PCA, the rand index increases, revealing a better understanding of the hidden structure of the data.

- **Number of clusters in relation to PCA dimension**



The information that can be retrieved from these graphs is:

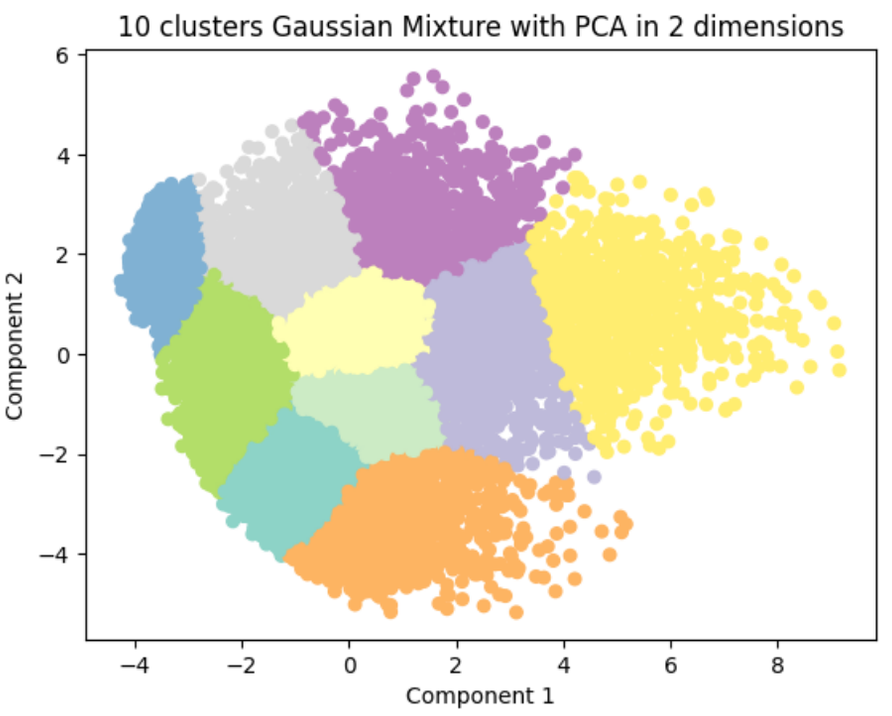
- The best executions are mostly characterised by 13 clusters;
- The worst executions are all characterised by 5 cluster.
- **Fitting and predicting time in relation to PCA dimension**

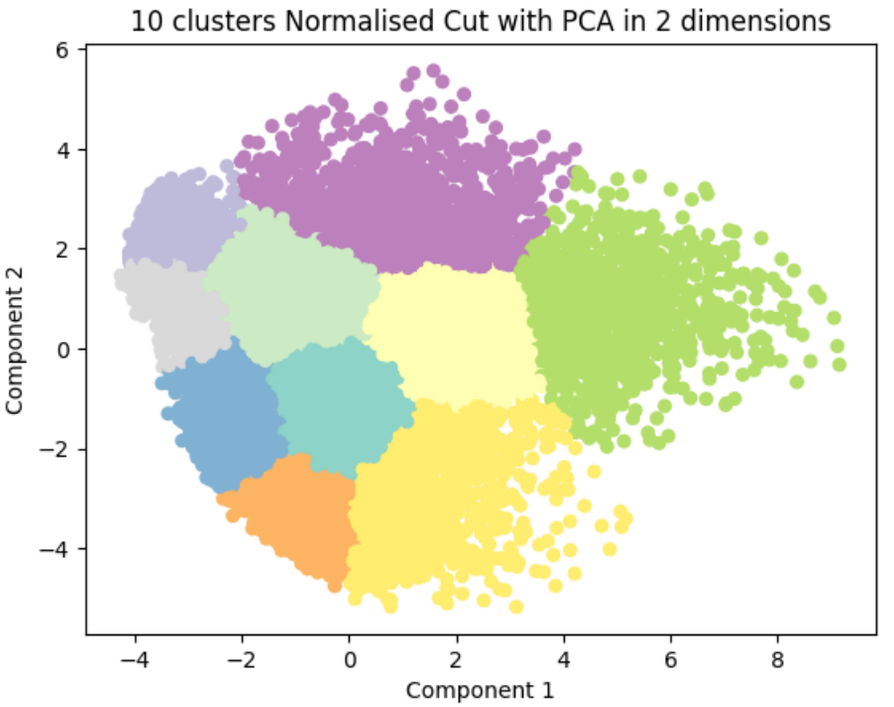


The weird thing that can be noticed is that the two trends are practically identical and the assumed behaviour says that by increasing the PCA the fitting and predicting time increases as well.
The times are not actually identical, they differ by thousandths of a second, however the underneath trends are the same.

Further Analysis

A further comparison can be done by visualizing the data with the best settings for the Gaussian Mixture Model and the Normalised Cut Model since in both the number of clusters can be made explicit; trivially, the number of selected clusters is 10 as the number of real labels:





Easy to see that the boundaries of the clusters are not exactly the same since the methods use two completely different approaches to solve the clustering problem, however they do not seem so different!

Conclusion

Let's better visualize the best settings of the presented models:

Model	PCA	Best hyper parameter 3	Rand score
Gaussian Mixture	22	15	0.90129
Mean Shift	22	5	0.908042
Normalised Cut	32	13	0.929651

The highest rand score is obtained by Normalised Cut, while the runner-up is Mean Shift with 0.908042. However, the number of clusters found by the algorithm far exceeds a reasonable number of possible clusters; indeed, it found 3524 clusters.

With regard to the rand index, the last model is the Gaussian Model, with a score equal to 0.90129. Nevertheless, it seems to find a good number of clusters with respect to the Mean Shift: only 15.

Model	Fit time	Predict time 3	Total time
Gaussian Mixture	0.469498	0.000000	0.469498
Mean Shift	175.233334	0.560470	175.793804
Normalised Cut	-	-	93.535461

With respect to the computational time, the fastest model is the Gaussian Mixture model with 0.469498 seconds, the second best is 93.535461 seconds and the worst in time is the Mean Shift model with 175.793804 seconds.

Taking into consideration both rand index and total execution time, the best ever model among all three is the Gaussian Mixture model:

- The rand index is good since it is over 90%;
- It's the fastest method with less than 1 second of execution.
- To perform well, it's sufficient a smaller number of PCA with respect to other methods and the number of clusters of the best settings is reasonable.

Bibliografia

[1] What is clustering? | IBM. URL https://www.ibm.com/topics/clustering.

[2] What is dimensionality reduction? | IBM. URL https://www.ibm.com/topics/dimensionality-reduction.

[3] Introduction to dimensionality reduction. URL https://www.geeksforgeeks.org/dimensionality-reduction/. Section: Machine Learning.

[4] The graph laplacian. URL https://mbernste.github.io/posts/laplacian_matrix/.

[5] Gaussian mixture model explained. URL https://builtin.com/articles/gaussian-mixture-model.

[6] Graph theory. URL https://en.wikipedia.org/w/index.php?title=Graph_theory&oldid=1243009263. Page Version ID: 1243009263.

[7] Kernel density estimation. URL https://en.wikipedia.org/w/index.php?title=Kernel_density_estimation&oldid=1228813181. Page Version ID: 1228813181.

[8] ML | mean-shift clustering. URL https://www.geeksforgeeks.org/ml-mean-shift-clustering/. Section: AI-ML-DS.

[9] Principal component analysis, . URL https://en.wikipedia.org/w/index.php?title=Principal_component_analysis&oldid=1232418328. Page Version ID: 1232418328.

[10] What is principal component analysis (PCA)? | IBM, . URL https://www.ibm.com/topics/principal-component-analysis.

[11] Rand index. URL https://en.wikipedia.org/w/index.php?title=Rand_index&oldid=1171913170. Page Version ID: 1171913170.

[12] Rand-index in machine learning. URL https://www.geeksforgeeks.org/rand-index-in-machine-learning/. Section: AI-ML-DS.

[13] Spectral clustering. URL https://en.wikipedia.org/w/index.php?title=Spectral_clustering&oldid=1242522903. Page Version ID: 1242522903.

[14] Unsupervised learning. URL https://en.wikipedia.org/w/index.php?title=Unsupervised_learning&oldid=1244566381. Page Version ID: 1244566381.

[15] What is unsupervised learning? | IBM. URL https://www.ibm.com/topics/unsupervised-learning.

[16] Jason Brownlee. A gentle introduction to expectation-maximization (EM algorithm). URL https://machinelearningmastery.com/expectation-maximization-em-algorithm/.

[17] Michael Greenacre, Patrick J. F. Groenen, Trevor Hastie, Alfonso Iodice D'Enza, Angelos Markos, and Elena Tuzhilina. Principal component analysis. 2(1):1–21. ISSN 2662-8449. doi: 10.1038/s43586-022-00184-w. URL https://www.nature.com/articles/s43586-022-00184-w. Publisher: Nature Publishing Group.

[18] Shreya Joshi. Types of clustering algorithms in machine learning with examples. URL https://www.analytixlabs.co.in/blog/types-of-clustering-algorithms/.

[19] Sanatan Mishra. Unsupervised learning and data clustering. URL https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a.

[20] Juan C. Olamendy. Understanding gaussian mixture models: A comprehensive guide. URL https://medium.com/@juanc.olamendy/understanding-gaussian-mixture-models-a-comprehensive-guide-df30af59ced7.

[21] Marcello Pelillo. *Clustering*.

[22] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation.

[23] Gilbert Tanner. Mean shift. URL https://ml-explained.com/blog/mean-shift-explained.

[24] Andrea Torsello. *Clustering*. .

[25] Andrea Torsello. *FeatureSelection*. .