Università
Ca'Foscari
Venezia

Artificial Intelligence and Data Engineering

Foundation of Artificial Intelligence

Assignment 3: **Clustering**

Professor
Andrea Torsello

Autor
Davide Tonetto
Student ID 884585

Academic year
2023/2024

Contents

List of Figures

Chapter 1

Introduction

1.1 Clustering assignment

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 assess the effect of the dimensionality on 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: means of the Gaussian models can be visualized as a greyscale image after PCA reconstruction to inspect the learned model.

1.2 Dataset

The MNIST dataset is composed of 70000 28x28 pixel gray-scale images of handwritten digits. The images are labeled with the corresponding digit.

Here are some examples of images in the dataset with their labels:

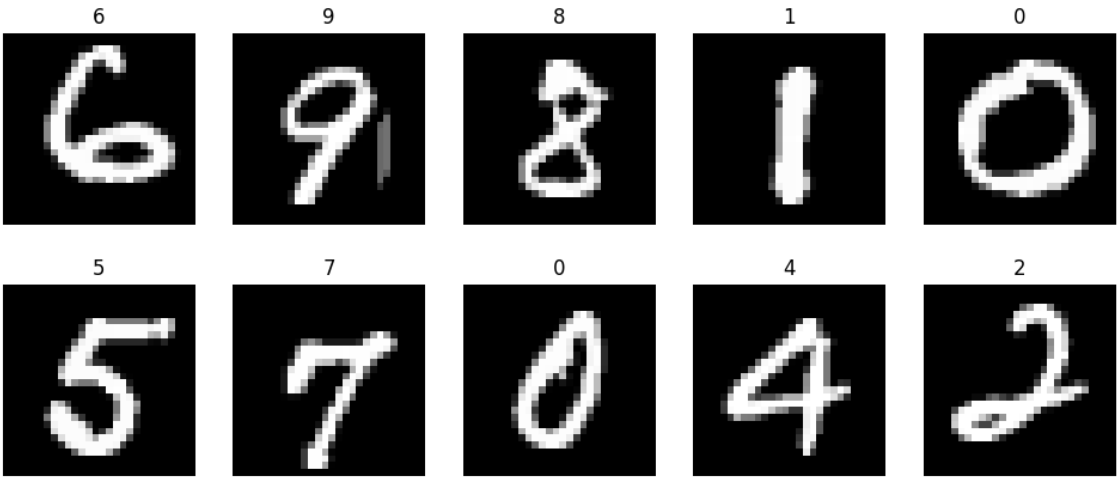

Figure 1.1: MNIST dataset examples.

1.3 Project structure

The project contains the following files:

- **data_engineering.ipynb:** Jupyter notebook containing the code to load the MNIST dataset and perform the reduction of the data through PCA.
- **GaussianMixture.ipynb:** Jupyter notebook containing the code to perform the classification using a mixture of Gaussians with diagonal covariance.
- **MeanShift.ipynb:** Jupyter notebook containing the code to perform the classification using mean shift.
- **NormalizedCut.ipynb:** Jupyter notebook containing the code to perform the classification using normalized cut.

Every one of the three notebooks used to perform clustering contains the code to perform the classification at varying levels of dimensionality reduction through PCA and presents the results through plots and tables.

Chapter 2

Unsupervised learning

2.1 Defining unsupervised learning

Unsupervised learning is a type of machine learning that looks for previously undetected patterns in a dataset with no pre-existing labels. It is used to draw inferences from datasets consisting of input data without labeled responses. The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data. Let's give a formal definition of unsupervised learning [1].

Definition 1 (Unsupervised learning)**.** *Given a dataset* $\mathcal{D} = \{x_1, x_2, \ldots, x_n\}$*, unsupervised learning is the task of learning a function* $f : \mathcal{X} \to \mathcal{Y}$ *that maps input data* $x \in \mathcal{X}$ *to an output* $y \in \mathcal{Y}$*, where* $\mathcal{Y}$ *is the set of possible outputs, without any supervision.*

In unsupervised learning, the goal is to find hidden structures in the data. This can be done in a variety of ways, such as finding the main patterns in the data, finding the main features of the data, or finding the main groups of the data. Two of the most common unsupervised learning methods are **clustering** and **dimensionality reduction**.

In Figure 2.1, we can see an example of unsupervised learning. We have a dataset of points in a 2D space, and we want to find the main groups of the data. In this case, we can use a clustering algorithm to find the main groups of the data.

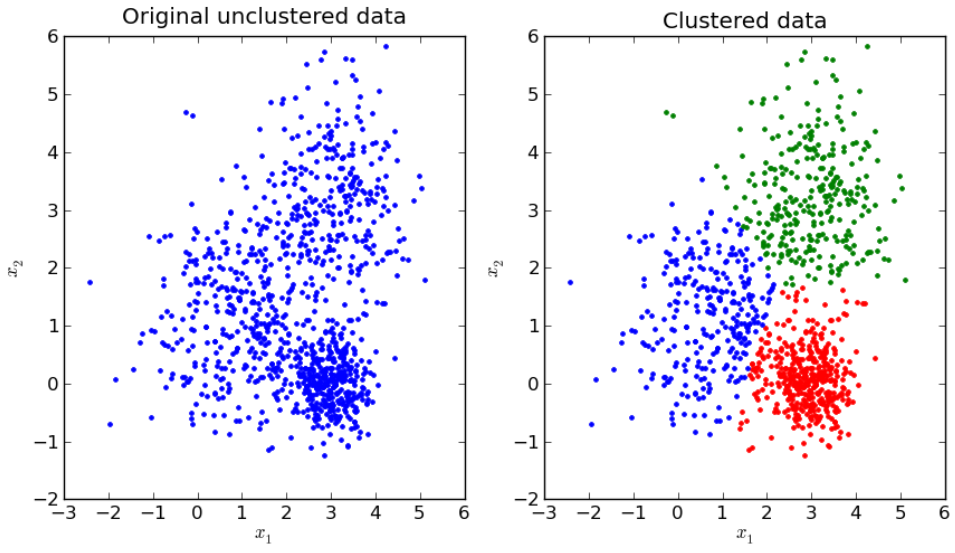

Figure 2.1: Example of unsupervised learning.

2.2 Clustering

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters). It is used in many fields such as machine learning, data mining, pattern recognition, image analysis, and bioinformatics. More formally clustering can be defined as follows.

Definition 2 (Clustering)**.** *Given a set of n objects and a nxn matrix A of pairwise similarities that give us an edge-weighted graph G. The goal of the clustering problem is to partition the vertices of G into k clusters such that the edges within clusters have high weights and the edges between clusters have low weights (maximally homogeneous groups) [4].*

There is a large variety of clustering algorithms, we will see three main types of clustering algorithms:

- **Distribution-based clustering**: These models are based on the assumption that the data is generated by a mixture of several distributions with different parameters. The most popular distribution-based clustering algorithm is the Gaussian Mixture Model (GMM).
- **Density-based clustering**: These models are based on the idea that a cluster is a dense area of the data space, separated by areas of lower density. The most popular density-based clustering algorithm is DBSCAN, but we'll see the Mean shift model in this text.
- **Graph-based clustering**: These models are based on the idea that the data can be represented as a graph, and the clusters are the connected components of the graph. We'll see the Normalized Cut algorithm in this text.

2.2.1 Rand index

The Rand index is a measure of the similarity between two data clusterings. It considers all pairs of elements and counts pairs that are either in the same cluster or in different clusters in both the true and predicted clusterings. The Rand index is defined as follows.

Definition 3 (Rand index)**.** *Given a set of n elements, the Rand index is given by the following formula:*

$$R = \frac{a+b}{\binom{n}{2}} = \frac{2(a+b)}{n(n-1)} \tag{2.1}$$

where a is the number of pairs of images that represent the same digit and that are clustered together and b is the number of pairs of images that represent different digits and that are placed in different clusters.

The Rand index always takes on a value between 0 and 1 where:

- 0: Indicates that two clustering methods do not agree on the clustering of any pair of elements.
- 1: Indicates that two clustering methods perfectly agree on the clustering of every pair of elements. [6]

2.3 Dimensionality reduction

Dimensionality reduction is the process of reducing the number of random variables under consideration by obtaining a set of principal variables. It can be divided into **feature selection** and **feature extraction**.

- **Feature selection**: Feature selection involves selecting a subset of the original features that are most relevant to the problem at hand. The goal is to reduce the dimensionality of the dataset while retaining the most important features. There are several methods for feature selection, including filter methods, wrapper methods, and embedded methods. Filter methods rank the features based on their relevance to the target variable, wrapper methods use the model performance as the criteria for selecting features, and embedded methods combine feature selection with the model training process.
- **Feature extraction**: Feature extraction involves creating new features by combining or transforming the original features. The goal is to create a set of features that captures the essence of the original data in a lower-dimensional space. There are several methods for feature extraction, including principal component analysis (PCA), linear discriminant analysis (LDA), and t-distributed stochastic neighbor embedding (t-SNE). PCA is a popular technique that projects the original features onto a lower-dimensional space while preserving as much of the variance as possible. [2]

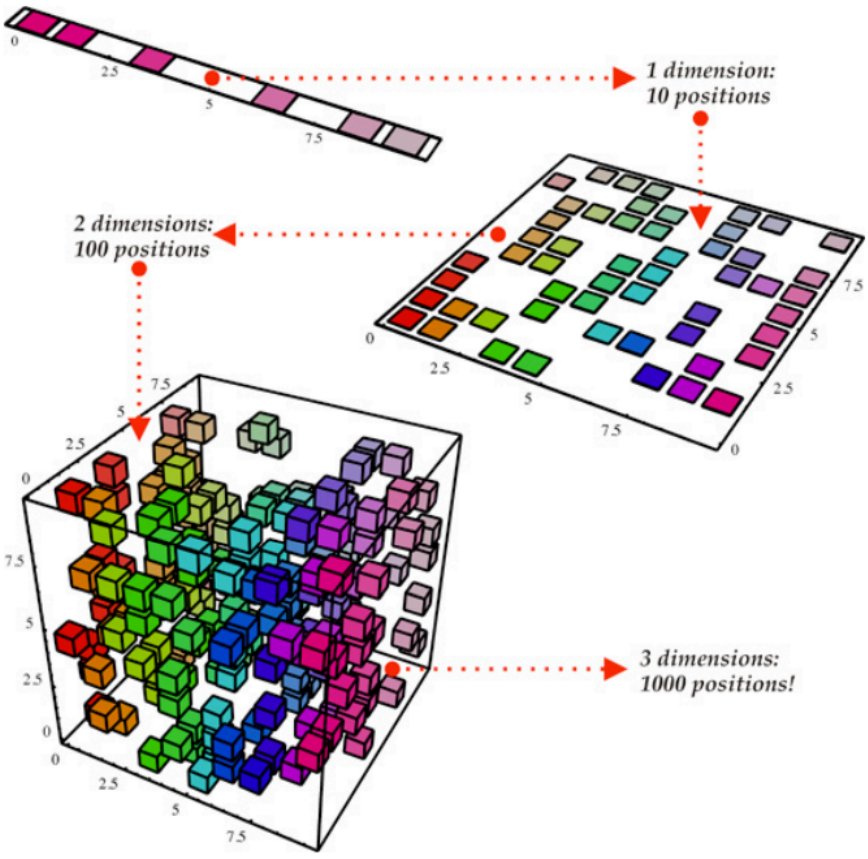

Figure 2.2: Example of dimensionality reduction.

2.3.1 Curse of dimensionality

The curse of dimensionality refers to the fact that many algorithms that work well in low dimensions become intractable when the input is high-dimensional. This is because the volume of the space increases exponentially with the number of dimensions, which makes it difficult to find a representative sample of the data. In addition, the distance between points becomes less meaningful in high dimensions, which makes it difficult to measure similarity.

Dimensionality reduction can help to reduce the computational cost and the risk of overfitting. There are two main types of dimensionality reduction: **linear** and **non-linear**. We will see the most popular linear dimensionality reduction algorithm, Principal Component Analysis (PCA).

2.3.2 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a linear dimensionality reduction technique that is widely used in machine learning. It works by finding the principal components of the data, which are the directions in which the data varies the most. These principal components can be used to project the data onto a lower-dimensional space while preserving as much of the variance as possible.

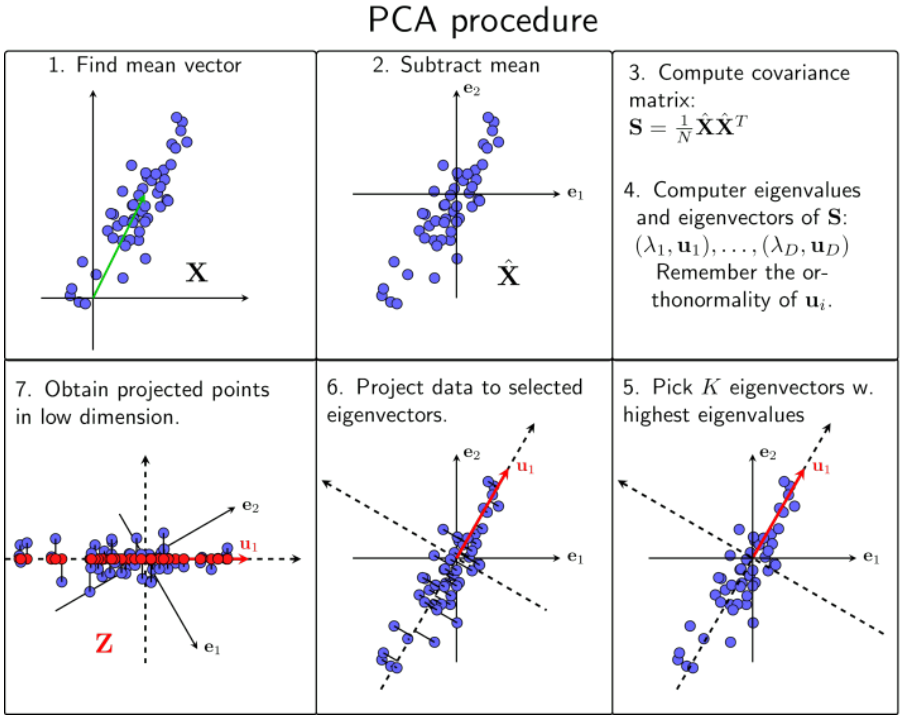

Figure 2.3: Principal Component Analysis procedure.

The principal components are the eigenvectors of the covariance matrix of the data, and the corresponding eigenvalues represent the amount of variance explained by each principal component. The first principal component is the direction in which the data varies the most, and the second principal component is the direction in which the data varies the second most, and so on.

Let's give a formal definition of PCA. Given a set of vectors $\{x_1, ..., x_n\}$ the first principal component is the unit vector μ that maximizes the variance of the projection of the data, where:

- The mean of the data along the direction of μ is $\mu^T \bar{x}$, where $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$.
- The variance of the projected data is:

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

  where S is the covariance matrix of the data:

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

Thus the variance is maximized by the unit vector μ that maximizes $\mu^T S \mu$ subject to the constraint $\mu^T \mu = 1$. This is a standard problem in linear algebra and it can be solved by finding the eigenvectors of the covariance matrix. The first principal component is the eigenvector corresponding to the largest eigenvalue of the covariance matrix, the second principal component is the eigenvector corresponding to the second largest eigenvalue, and so on. [1]

2.3.3 Applying PCA to the MNIST dataset

For the aim of the project, we will apply PCA to the MNIST dataset. In the **data_egineering.ipynb** file, we will load the MNIST dataset and apply PCA to it (from 2 to 200 components). We will then visualize the first two principal components of the data.

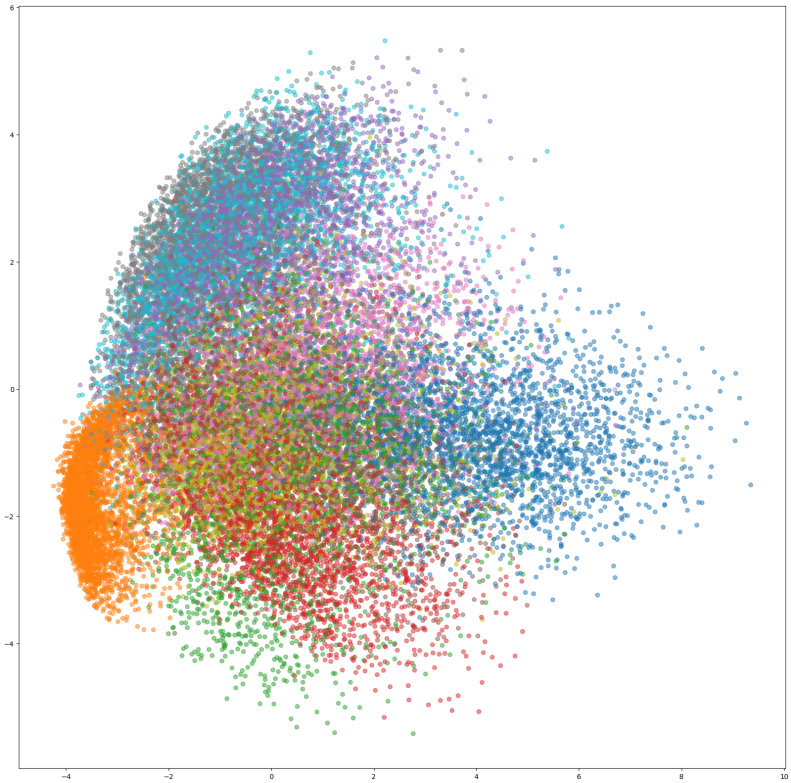

Figure 2.4: First two principal components of the MNIST dataset.

The PCA is applied using the **sklearn** library. The **PCA** class is used to apply PCA to the data. The **fit_transform** method is used to fit the PCA model to the data and transform the data into the lower-dimensional space. The **explained_variance_ratio_** attribute is used to get the explained variance ratio of each principal component.

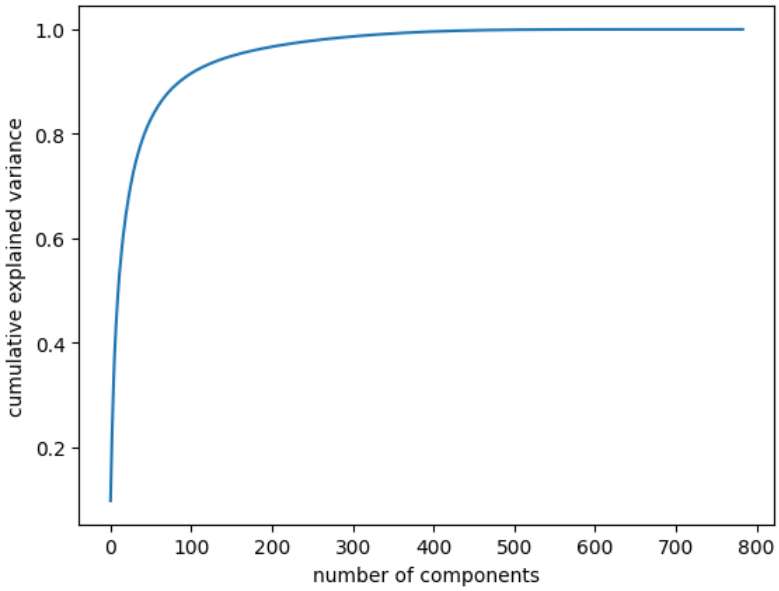

Figure 2.5: Cumulative explained variance of the PCA.

The cumulative explained variance of the PCA is shown in the figure above. The cumulative explained variance is the sum of the explained variance ratio of each principal component. It represents the amount of variance explained by the first n principal components. We can see that the first two principal components explain about 16.9% of the variance, and the first 200 principal components explain about 96.7% of the variance.

Saving PCA from 2 to 200 components

In the **data_egineering.ipynb** file, we will save the PCA from 2 to 200 components using the following Python code:

```
# Find a subset of indexes for the X with a seed
np.random.seed(42)
subset_indexes = np.random.choice(X.index, 50000, replace=False)

X_subset = X.loc[subset_indexes]
y_subset = y.loc[subset_indexes]

# save the subset data to a file
X_subset.to_feather('./data/mnist_subset.feather')
y_subset.to_feather('./data/mnist_subset_labels.feather')

for i in tqdm.tqdm(range(2, 201)):
    pca = PCA(n_components=i)
```

```
    pca.fit(X)

    X_pca = pca.transform(X_subset)
    X_pca = pd.DataFrame(X_pca)

    # save the PCA data and obj to a file
    X_pca.to_feather(f'./data/pca/mnist_pca_{i}.feather')
    with open(f'./models/pca/mnist_pca_{i}.pkl', 'wb') as f:
        pickle.dump(pca, f)
```

Initially, we find a subset of indexes for the X with a seed in order to reduce the execution time for the models that we will see later. We then save the subset data to a file. We then loop from 2 to 200 and apply PCA to the subset data. We then save the PCA data and object to a file.

Chapter 3

Gaussian Mixture

3.1 Defining Gaussian Mixture Model

A Gaussian Mixture Model (GMM) is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians. In the simplest case, GMMs can be used for finding clusters in the same manner as k-means.

3.1.1 GMM in unsupervised clustering

We are talking about **unsupervised clustering** which is the problem of discerning multiple categories in a collection of objects (the problem is unsupervised because the category labels are not given). Unsupervised clustering begins with data (Figure 3.1-(b)) next, we have to understand what kind of probability distribution the data follows. In the case of GMM, we assume that the data follows a Gaussian distribution. Clustering presumes that the data are generated from a mixture distribution (P). Such a distribution has k components, each of which is a distribution in its own right. A data point is generated by first choosing a component and then generating a sample from that component. Let the random variable C denote the component, with values $1, ..., k$; then the mixture distribution is given by:

$$P(x) = \sum_{i=1}^{k} P(C = i)P(x|C = i)$$

where x refers to the values of the attributes for a data point. For continuous data, a natural choice for the component distributions is the multivariate Gaussian, which gives the so-called mixture of Gaussian family of distributions [5].

the parameters of the model are the component weights, the means, and the covariance matrices. The component weights, denoted by π_i, are non-negative and sum to 1. The means, denoted by μ_i, are the means of the component distributions. The covariance matrices, denoted by Σ_i, are the covariance matrices of the component distributions. The likelihood of a data point x is given by:

$$P(x) = \sum_{i=1}^{k} \pi_i \mathcal{N}(x|\mu_i, \Sigma_i)$$

where $\mathcal{N}(x|\mu_i, \Sigma_i)$ is the Gaussian distribution for component i. The parameters

of the model are usually estimated by maximum likelihood, using the Expectation-Maximization (EM) algorithm.

3.1.2 Expectation-Maximization Algorithm

The EM algorithm is an iterative method that alternates between performing an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the E step. The algorithm is guaranteed to converge to a local maximum of the likelihood function.

For the mixture of Gaussians, we initialize the mixture-model parameters arbitrarily and then iterate the following two steps until convergence:

1. **E-step**: Compute the probabilities $p_{ij} = P(C = i|x_j)$, the probability that x_j was generated by component i. By Bayes' theorem, we have $p_{ij} = \alpha P(x_j|C = i)P(C = i)$. The term $P(x_j|C = i)$ is just the probability at x_j of the i-th Gaussian, and $P(C = i)$ is the weight of the i-th Gaussian. Define $n_i = \sum_j p_{ij}$, the effective number of points assigned to the i-th Gaussian.

2. **M-step**: Compute the new mean, covariance, and component weights using the following steps in sequence:

   (a) The new mean is $\mu_i = \frac{1}{n_i} \sum_j p_{ij} x_j$.

   (b) The new covariance is $\Sigma_i = \frac{1}{n_i} \sum_j p_{ij}(x_j - \mu_i)(x_j - \mu_i)^T$.

   (c) The new component weight is $\pi_i = \frac{n_i}{n}$, where n is the total number of points.

The final model that EM learns when it is applied to the data in Figure 3.1-(a) is shown in Figure 3.1-(c); it is virtually indistinguishable from the original model from which the data were generated. Figure 3.1-(a) plots the log-likelihood of the data according to the current model as EM progresses [5].

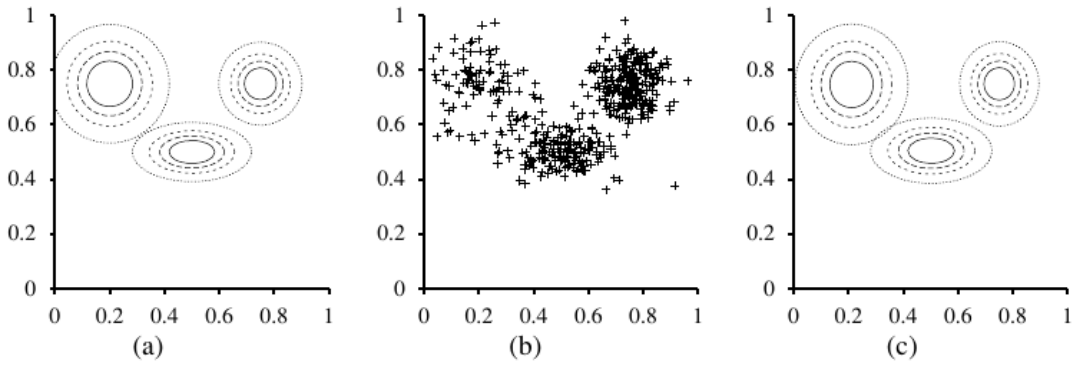

Figure 3.1: (a) A Gaussian mixture model with three components; the weights (left-to-right) are 0.2, 0.3, and 0.5. (b) 500 data points sampled from the model in (a). (c) The model reconstructed by EM from the data in (b). [5]

3.2 GMM with MNIST dataset

In this section, we will apply the Gaussian Mixture Model to the MNIST dataset. The library used for this purpose is `scikit-learn` and the parameters of the model are the following:

- **n_components**: the number of components in the mixture.
- **covariance_type**: the type of covariance parameters to use. The parameter can be one of the following:
  - `full`: each component has its own general covariance matrix.
  - `tied`: all components share the same general covariance matrix.
  - `diag`: each component has its own diagonal covariance matrix.
  - `spherical`: each component has its own single variance.
- **max_iter**: the number of EM iterations to perform.

The parameters used for the GMM are the following:

- **n_components**: from 5 to 15
- **covariance_type**: `diag`
- **max_iter**: 1000

Also, the model has been applied on data obtained from the application of PCA to the MNIST dataset using from 2 to 200 components.

3.2.1 Outputs

The following table shows the results obtained from the application of the GMM on the MNIST dataset using the parameters described above. The table shows the score, the fit time, and the predict time for each number of components used, and for each row, the number of components (from 5 to 15) that gave the best score is shown.

Table 3.1: GMM with MNIST dataset results

PCA	n. components	score	fit time	predict time
2	14	0.873063	0.952908	0.038013
3	15	0.874683	1.265872	0.054549
4	15	0.886725	1.108679	0.074184
5	15	0.880446	1.811385	0.038814
6	15	0.888323	1.564815	0.037781
7	15	0.891685	2.243591	0.042535
8	15	0.891844	2.602756	0.040036
9	15	0.886491	1.676407	0.041473
10	15	0.893340	2.028031	0.041598
11	15	0.894598	1.420239	0.042014
12	15	0.890592	2.599520	0.040164
13	15	0.893809	2.793790	0.042342

Continued on next page

Table 3.1 – *Continued from previous page*

PCA	n. components	score	fit time	predict time
14	15	0.897146	2.518098	0.041817
15	15	0.888423	2.287862	0.043640
16	15	0.895279	4.011619	0.034789
17	15	0.899635	1.752979	0.032549
18	15	0.898451	2.535492	0.047873
19	15	0.891675	4.231398	0.033083
20	15	0.896705	2.924212	0.033687
21	15	0.890761	2.558386	0.044874
22	15	0.898398	3.109447	0.048259
23	15	0.887768	2.818469	0.043546
24	14	0.891575	2.445534	0.041455
25	14	0.899274	2.397341	0.041007
26	15	0.893884	2.350519	0.048223
27	14	0.892834	2.777335	0.044442
28	15	0.897999	2.702518	0.040426
29	15	0.898225	3.273701	0.046145
30	13	0.890508	3.541791	0.044157
31	15	0.896943	2.965677	0.046391
32	14	0.885087	3.458991	0.046970
33	15	0.884797	3.195530	0.047147
34	15	0.884793	4.419576	0.050545
35	15	0.892055	3.913225	0.045787
36	13	0.881255	2.393417	0.131871
37	15	0.893279	2.797726	0.047109
38	15	0.895572	2.591986	0.046597
39	15	0.888542	3.883247	0.047360
40	14	0.893250	3.298369	0.043265
41	15	0.876372	9.326422	0.047402
42	15	0.860792	4.324344	0.048883
43	11	0.859821	4.625269	0.033558
44	13	0.866274	2.849989	0.046395
45	14	0.885879	2.666476	0.047631
46	13	0.890746	2.995172	0.049429
47	15	0.871273	6.104545	0.050229
48	13	0.887287	3.202717	0.040752
49	15	0.857056	4.646554	0.053224
50	14	0.873953	6.628042	0.039479
51	13	0.877326	3.276040	0.038550
52	12	0.886495	2.617009	0.056954
53	15	0.864515	5.839053	0.045053
54	12	0.878970	3.737827	0.037782
55	13	0.874804	3.387508	0.041792
56	15	0.853941	7.800612	0.043514
57	15	0.866737	5.358308	0.050644
58	12	0.876900	3.431551	0.054922

Continued on next page

Table 3.1 – *Continued from previous page*

PCA	n. components	score	fit time	predict time
59	15	0.879948	6.741478	0.042735
60	9	0.851839	2.200061	0.040292
61	15	0.850233	7.903400	0.042996
62	15	0.857269	6.982940	0.047153
63	15	0.855870	3.428830	0.044937
64	13	0.870841	3.569763	0.041489
65	15	0.869254	3.488273	0.053347
66	15	0.858357	4.238203	0.046170
67	13	0.861852	3.998077	0.041141
68	13	0.859214	3.172544	0.043043
69	15	0.854765	4.820639	0.045613
70	15	0.846959	5.134275	0.044021
71	15	0.852286	3.248270	0.046840
72	15	0.850810	8.111901	0.046682
73	12	0.848869	2.954851	0.049103
74	11	0.846888	3.389351	0.045776
75	12	0.850320	3.028174	0.045042
76	15	0.853165	3.675785	0.051715
77	14	0.849658	5.092500	0.054749
78	15	0.856429	5.608951	0.048193
79	14	0.847904	8.220242	0.046937
80	13	0.855019	4.329116	0.046463
81	15	0.851539	3.624961	0.049022
82	15	0.852901	8.447213	0.052682
83	14	0.846660	3.895931	0.048004
84	15	0.842960	3.981878	0.051348
85	15	0.857434	11.094822	0.051069
86	10	0.845340	2.769198	0.042703
87	14	0.867776	4.047026	0.050927
88	14	0.846460	4.376738	0.050222
89	14	0.856539	4.672426	0.052442
90	14	0.863298	3.826893	0.049754
91	14	0.838898	9.134682	0.049689
92	13	0.834866	3.106788	0.050882
93	14	0.848204	4.041180	0.051780
94	15	0.851977	9.090871	0.056013
95	15	0.854780	7.710920	0.055252
96	15	0.850358	4.246728	0.051190
97	13	0.851978	4.181516	0.051295
98	13	0.855860	3.908176	0.047988
99	15	0.852604	4.620289	0.055549
100	14	0.833796	9.831733	0.058767
101	15	0.839464	5.541362	0.058688
102	15	0.839601	5.367220	0.058981
103	13	0.848557	3.711401	0.052262

Continued on next page

Table 3.1 – *Continued from previous page*

PCA	n. components	score	fit time	predict time
104	15	0.853247	6.994402	0.060553
105	13	0.860647	6.876982	0.055189
106	15	0.850634	9.745014	0.063364
107	15	0.826030	10.002733	0.057491
108	15	0.840798	6.472421	0.054521
109	11	0.838067	4.154619	0.062586
110	15	0.855307	8.052541	0.054804
111	15	0.837057	5.691788	0.060870
112	11	0.845053	4.807609	0.055938
113	15	0.842759	4.682725	0.056625
114	15	0.846878	6.931569	0.113605
115	15	0.860502	8.271335	0.066541
116	12	0.830946	3.929992	0.063018
117	12	0.839188	5.736313	0.064054
118	14	0.850398	5.427718	0.070079
119	15	0.833457	6.144699	0.068853
120	12	0.844927	4.007333	0.064915
121	13	0.861106	5.697190	0.077950
122	14	0.828984	11.097002	0.067895
123	15	0.856462	8.340294	0.078401
124	15	0.840303	6.523578	0.075737
125	14	0.854308	4.569740	0.065916
126	14	0.847503	7.511065	0.069499
127	15	0.845710	7.923840	0.070621
128	14	0.836971	7.296070	0.066115
129	14	0.836830	6.611978	0.066745
130	14	0.847519	7.140758	0.072519
131	13	0.824597	8.405560	0.066953
132	14	0.830171	10.508137	0.067903
133	15	0.846633	5.672877	0.068761
134	12	0.833481	5.250885	0.067474
135	15	0.827271	11.428955	0.069754
136	11	0.829320	3.757688	0.069724
137	14	0.829074	9.879141	0.085655
138	12	0.833963	3.432525	0.140726
139	14	0.830015	8.859976	0.067677
140	13	0.847416	4.454234	0.072333
141	14	0.829346	4.932285	0.077629
142	12	0.820934	3.920100	0.088401
143	15	0.848727	9.609901	0.083089
144	10	0.825990	4.369861	0.066110
145	13	0.821218	8.460212	0.074736
146	15	0.846078	8.103657	0.074507
147	13	0.847255	5.825136	0.067432
148	14	0.828940	8.320999	0.072488

Continued on next page

Table 3.1 – *Continued from previous page*

PCA	n. components	score	fit time	predict time
149	11	0.835741	3.845506	0.071651
150	13	0.847111	5.190373	0.072841
151	14	0.846787	11.767635	0.077105
152	15	0.835817	10.430440	0.075171
153	14	0.839568	9.435308	0.070482
154	13	0.848115	5.597801	0.070080
155	15	0.840931	6.782761	0.080780
156	15	0.826009	15.036724	0.079066
157	14	0.840844	10.140429	0.078519
158	15	0.823731	7.485414	0.086569
159	15	0.823909	13.419016	0.084721
160	14	0.826837	10.037580	0.074829
161	13	0.844181	5.796721	0.077455
162	13	0.847067	5.392274	0.083917
163	15	0.841022	5.753653	0.075663
164	13	0.846492	6.209195	0.075433
165	15	0.818691	8.590430	0.078006
166	14	0.827457	12.356487	0.109448
167	14	0.826310	9.918793	0.088732
168	10	0.824827	4.283140	0.080311
169	15	0.825491	9.435661	0.080167
170	15	0.835786	10.836362	0.079623
171	11	0.827070	4.354459	0.089414
172	15	0.836686	8.319697	0.083155
173	12	0.837363	6.927452	0.079190
174	15	0.830476	11.789310	0.078768
175	15	0.826866	16.099412	0.154963
176	10	0.826595	3.972378	0.083466
177	13	0.826516	7.156279	0.081951
178	10	0.824155	4.021216	0.074246
179	15	0.820152	9.362758	0.083351
180	13	0.827017	6.003835	0.079543
181	12	0.841469	6.038951	0.082942
182	15	0.827795	13.727554	0.085765
183	11	0.828623	4.827980	0.090687
184	12	0.823882	6.656589	0.090960
185	13	0.813695	5.538610	0.099276
186	12	0.825755	6.332991	0.090019
187	12	0.812437	6.169416	0.112386
188	13	0.843072	6.799785	0.093233
189	9	0.808484	4.999704	0.078122
190	14	0.823769	15.060336	0.084549
191	12	0.827281	6.209378	0.109759
192	15	0.824883	13.890513	0.090327
193	12	0.813672	7.342996	0.085231

Continued on next page

Table 3.1 – *Continued from previous page*

PCA	n. components	score	fit time	predict time
194	15	0.825286	15.442775	0.087971
195	15	0.827180	9.845559	0.105245
196	15	0.836078	8.066704	0.115211
197	15	0.820875	10.717594	0.096686
198	15	0.825924	11.968893	0.100151
199	12	0.814362	5.719073	0.092225
200	15	0.831733	9.857435	0.088675

Overall, the best score obtained is the following:

- **PCA**: 17
- **n. components**: 15
- **Score**: 0.899635
- **Fit time**: 1.752979
- **Predict time**: 0.032549

3.2.2 Analysis

The following plots show respectively the best score related to the PCA dimension and the number of components used.

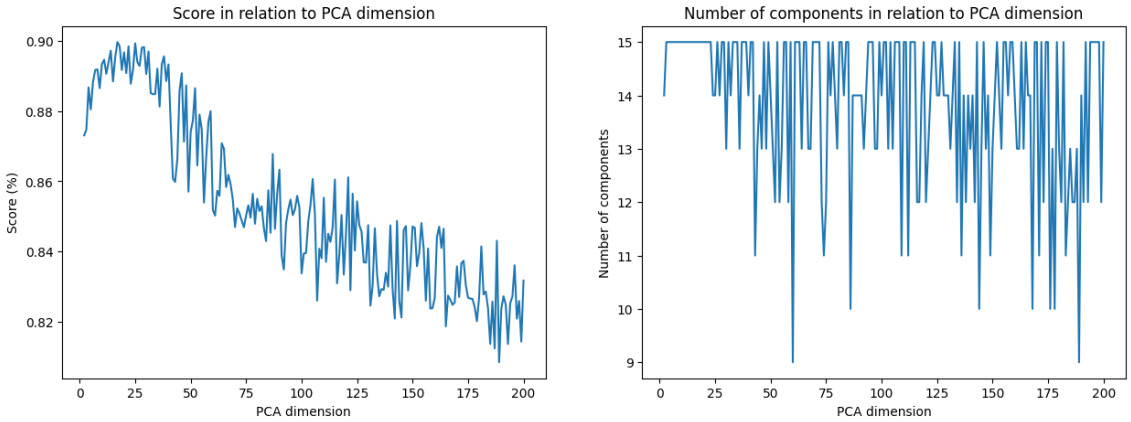

Figure 3.2: Score analysis of the GMM with MNIST dataset in relation to the PCA dimension

As we can see from the first plot in Figure 3.2, the score decreases as the PCA dimension increases. This is due to the fact that the model is not able to capture the variance of the data when the number of components is high. On the other hand, the number of components related to the best score is shown in the second plot in Figure 3.2 and we can see that is usually around 14.

The next plots show the fit time and the predict time related to the PCA dimension for the best scores obtained.

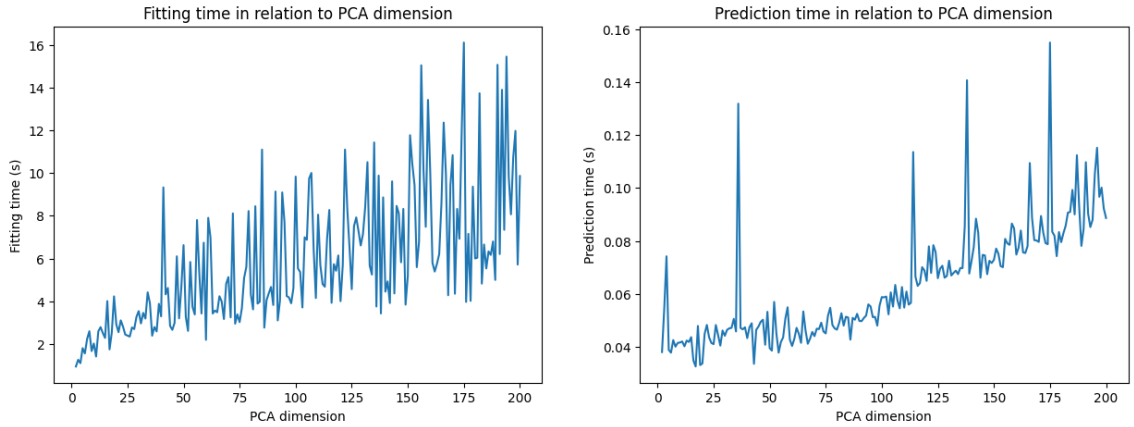

Figure 3.3: Time analysis of the GMM with MNIST dataset in relation to the PCA dimension

As we can see from the first plot in Figure 3.3, the fit time increases as the PCA dimension increases. Also the predict time increases as the PCA dimension increases, as shown in the second plot in Figure 3.3. Overall, the time is not high and the model is able to predict the data in a short time.

The next plot show the cluster in 2 dimensions and also the ellipses that represent the covariance of the clusters.

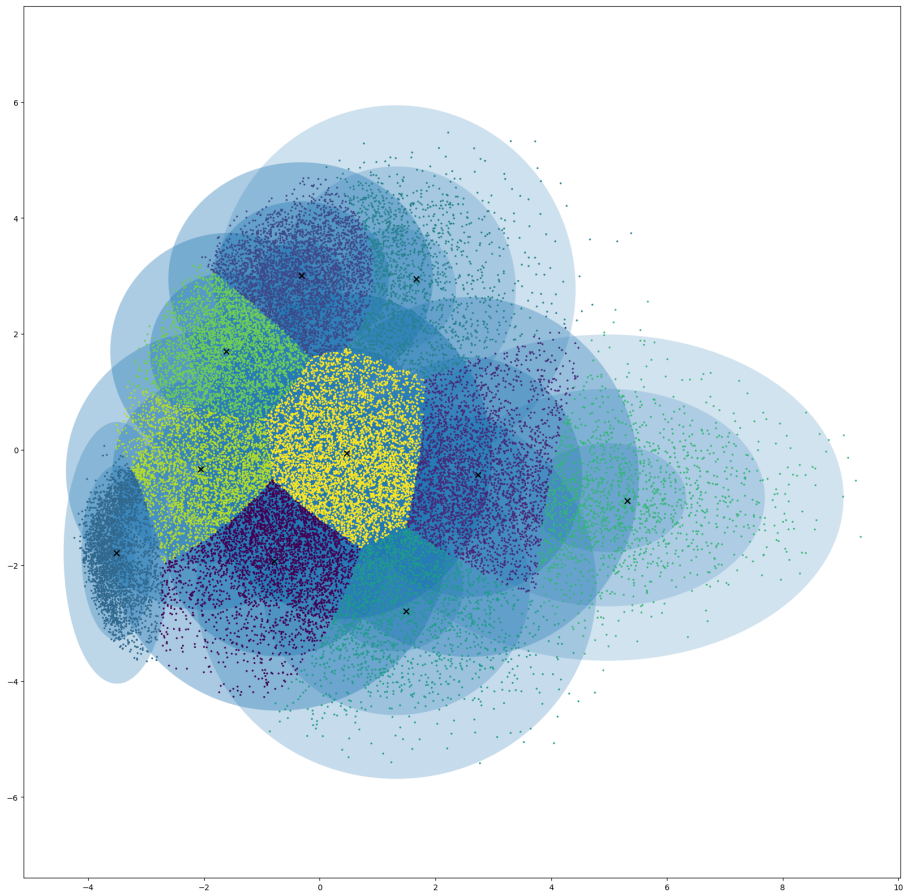

Figure 3.4: Cluster analysis of the GMM with MNIST dataset in 2 dimensions

Let's show now the means plots for the best and the worst score obtained.

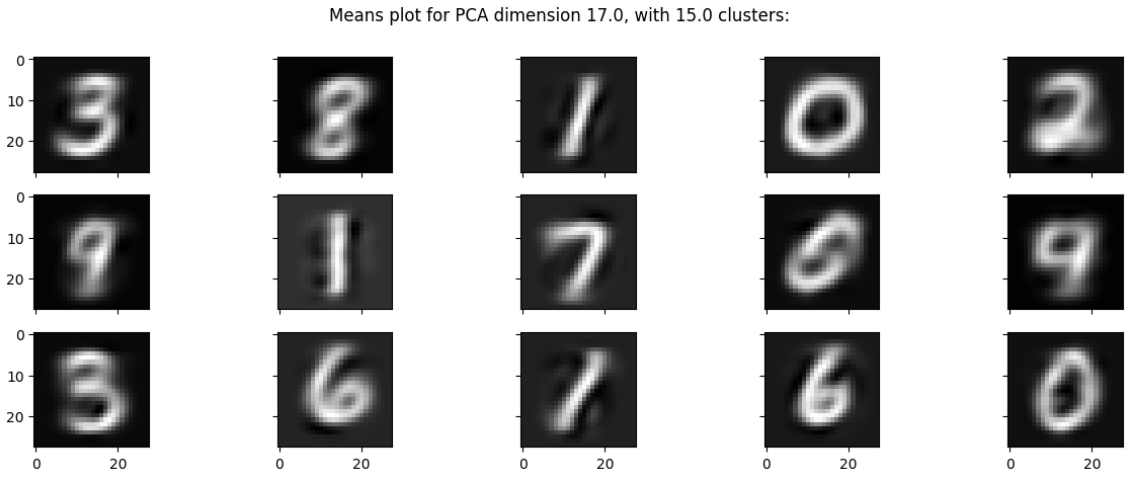

Figure 3.5: Means analysis of the GMM with MNIST dataset for the best score

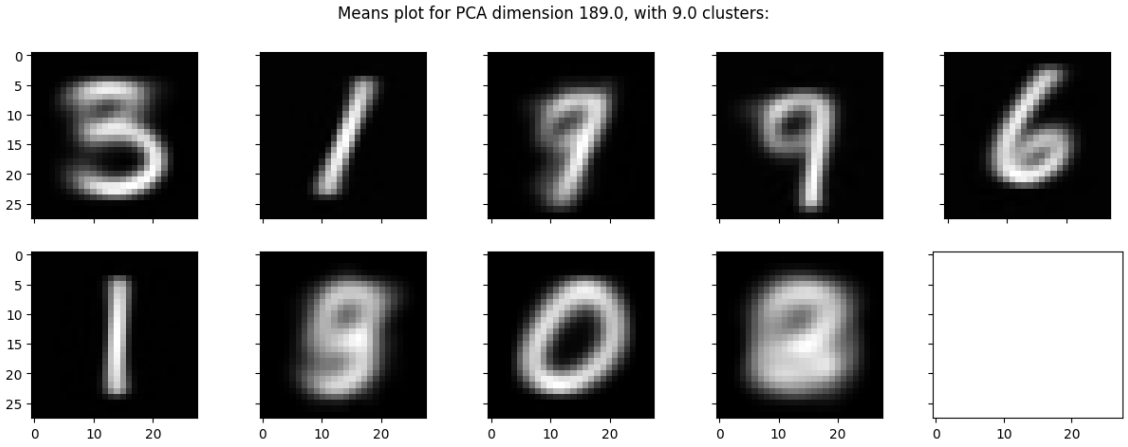

Figure 3.6: Means analysis of the GMM with MNIST dataset for the worst score

As we can see from the plots in Figure 3.5 and Figure 3.6, the means of the clusters are different for the best and the worst score obtained, the best score has a more defined cluster than the worst score and we can clearly distinguish the number represented by the cluster.

Chapter 4

Mean Shift

4.1 Defining Mean Shift

Mean shift is a non-parametric feature-space analysis technique for locating the maxima of a density function, a so-called mode-seeking algorithm. It is a procedure for locating the maxima of a density function given discrete data sampled from that function. It is useful for visualizing clusters in high-dimensional data, and it is also a method of interest for image processing. It has applications in computer vision and image processing. The algorithm is an iterative procedure that shifts each data point to the average of data points in its neighborhood.

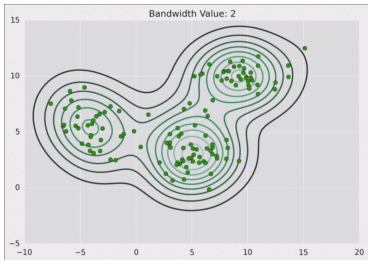

Figure 4.1: Mean Shidt - Contour plot a

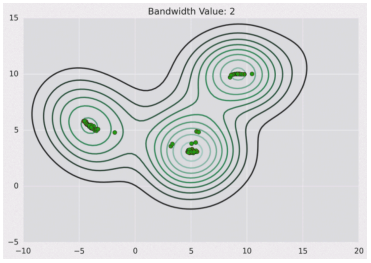

Figure 4.2: Mean Shidt - Contour plot b

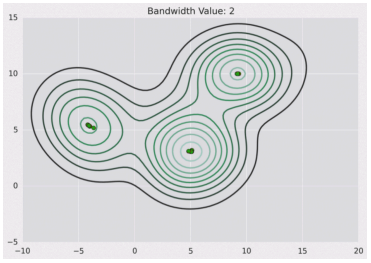

Figure 4.3: Mean Shidt - Contour plot c

More formally, Mean Shift is defined as follows:

Definition 4 (Mean Shift)**.** *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. So each iteration each data point will move closer to where the most points are at, which is or will lead to the cluster center. When the algorithm stops, each point is assigned to a cluster. [3]*

Unlike K-means, Mean Shift does not require the number of clusters to be determined in advance. The number of clusters is determined by the algorithm based on the density of the data points. One of the main advantages of Mean Shift is that it does not assume that the clusters are convex-shaped. This means that it can find clusters of any shape. However, the main disadvantage of Mean Shift is that it is computationally expensive, especially when the dataset is large ($O(n^2)$).

4.1.1 Mean Shift algorithm

The mean shift algorithm is a mode-seeking algorithm that assigns each data point to the nearest mode of the underlying density function. The algorithm is defined as follows:

1. **Initialization**: The algorithm starts by initializing a window around each data point. The window is defined by a bandwidth parameter h. The window can be a hypercube or a hypersphere, depending on the dimensionality of the data. The bandwidth parameter h determines the size of the window. A small value of h will result in a small window, and a large value of h will result in a large window.
2. **Mean Shift**: For each data point, the algorithm computes the mean of the data points that are within the window. The mean is computed using a kernel function. The kernel function is a function that assigns a weight to each data point based on its distance from the current data point. The weight decreases as the distance increases. The mean of the data points is computed as a weighted average, where the weight of each data point is determined by the kernel function. The mean shift vector is the difference between the current data point and the mean of the data points. The mean shift vector indicates the direction in which the data point should move to reach the mode of the underlying density function.
3. **Update**: The algorithm updates each data point by shifting it in the direction of the mean shift vector. The amount of shift is determined by the mean shift vector and a step size parameter. The step size parameter determines how far the data point should move in the direction of the mean shift vector. The algorithm repeats steps 2 and 3 until convergence. Convergence occurs when the mean shift vectors become small, indicating that the data points have reached the modes of the underlying density function.

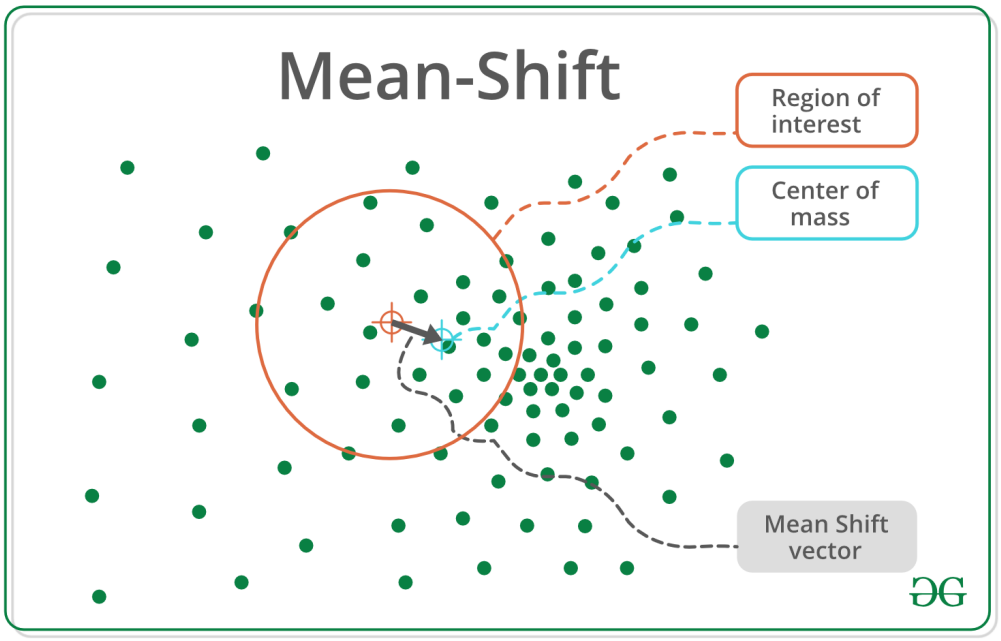

Figure 4.4: Mean Shift algorithm.

4.1.2 Kernel density estimation

The mean shift algorithm is based on kernel density estimation, which is a non-parametric method for estimating the probability density function of a random variable. It works by placing a kernel on each point in the data set. A kernel is a fancy

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

To illustrate, suppose we are given a data set $\{u_i\}$ of points in d-dimensional space, sampled from some larger population, and that we have chosen a kernel K having bandwidth parameter h. Together, these data and kernel function returns the following kernel density estimator for the full population's density function:

$$\hat{f}_k(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{u - u_i}{h}\right)$$

where K is the kernel function, h is the bandwidth parameter, and n is the number of data points. The kernel function K is a non-negative function that integrates to one. The bandwidth parameter h determines the size of the kernel. A small value of h will result in a narrow kernel, and a large value of h will result in a wide kernel. The kernel function assigns a weight to each data point based on its distance from the current data point. The weight decreases as the distance increases.

A popular kernel function is the Gaussian kernel. The Gaussian kernel is defined as:

$$K(u) = \frac{1}{(2\pi)^{d/2}} e^{-\frac{1}{2}|u|^2}$$

where u is the distance between the current data point and the data point being evaluated.

4.2 Mean Shift with MNIST dataset

In this section, we will apply the Mean Shift algorithm to the MNIST dataset. The library used for this purpose is `scikit-learn` and the parameters of the model are the following:

- `bandwidth`: The bandwidth parameter h of the kernel. The bandwidth parameter determines the size of the kernel. A small value of h will result in a narrow kernel, and a large value of h will result in a wide kernel. The default value is 1.0.
- `bin_seeding`: If true, initial kernel locations are not locations of all points, but rather the location of the discretized version of the data using a binning technique. The default value is False.
- `cluster_all`: If true, all points are clustered, even those orphans that are not within any kernel. The default value is True.
- `min_bin_freq`: The minimum number of points in a bin for it to be considered in the kernel density estimation. The default value is 1.

- `max_iter`: The maximum number of iterations to perform. The default value is 300.

The parameters used for the Mean Shift algorithm are the following:

- `bandwidth`: following values are tested: $[0.1, 0.5, 1, 2, 3, 4, 5, 6]$
- `bin_seeding`: default,
- `cluster_all`: default,
- `min_bin_freq`: default,
- `max_iter`: default.

Also, the model has been applied on data obtained from the application of PCA to the MNIST dataset using from 2 to 192 (interleaving 10) components.

4.2.1 Outputs

The following table shows the results obtained from the application of the Mean shift on the MNIST dataset using the parameters described above. The table shows the bandwidth, the number of clusters, the rand score, the fit time, and the predict time. This is for the best result of each number of components used in the PCA.

Table 4.1: Mean shift with MNIST dataset results

PCA	bandwidth	n. clusters	score	fit time	predict time
2	0.100000	3077	0.899633	85.289198	2.703851
12	2.000000	6370	0.903977	296.984033	5.217837
22	3.000000	4552	0.905631	763.364621	3.950438
32	4.000000	1475	0.904376	961.140019	1.374306
42	4.000000	3793	0.907117	1291.102779	3.475688
52	4.000000	6481	0.906542	1324.836578	3.838357
62	4.000000	9209	0.905987	1745.046934	5.328928
72	4.000000	11759	0.905586	1557.784941	7.315098
82	4.000000	13943	0.905105	1397.667253	8.689452
92	5.000000	3229	0.905372	2493.033968	3.952580
102	5.000000	3822	0.906390	2718.879880	3.784310
112	5.000000	4437	0.906411	2940.073711	4.476541
122	5.000000	5012	0.906621	3313.830433	3.225683
132	5.000000	5507	0.906784	3558.628960	3.437061
142	5.000000	6050	0.906777	3746.892362	3.780931
152	5.000000	6506	0.906895	3855.355446	4.171000
162	5.000000	6972	0.907053	3819.870814	4.706242
172	5.000000	7406	0.906976	4064.102257	4.759609
182	5.000000	7806	0.906960	4140.518807	4.956070
192	5.000000	8176	0.907037	4697.245787	5.497378

Overall, the best results are obtained with 42 components and a bandwidth of 4.0. The number of clusters is 3793, the rand score is 0.907117, the fit time is 1291.102779 seconds, and the predict time is 3.475688 seconds.

4.2.2 Analysis

The following plots show respectively the best score related to the PCA dimension and the bandwidth used.

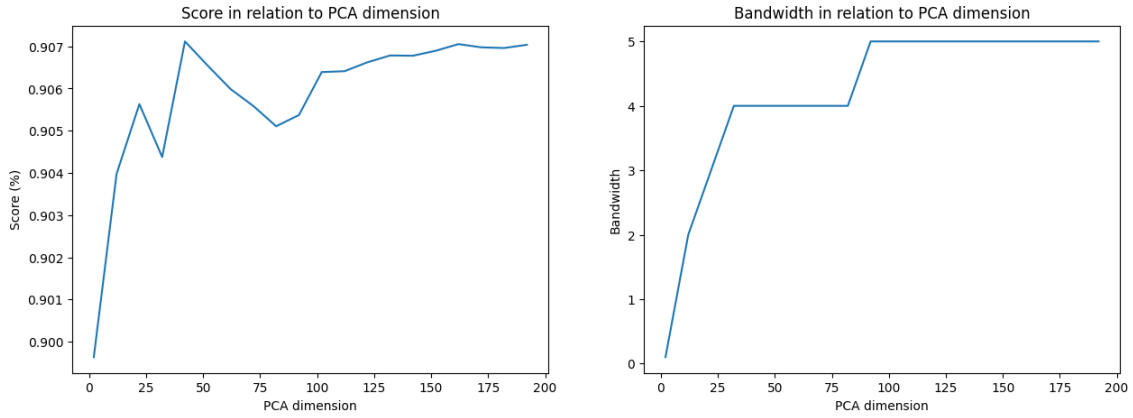

Figure 4.5: Score analysis of the Mean shift with MNIST dataset in relation to the PCA dimension

As we can see from the first plot the score increases with the number of components used in the PCA. This is due to the fact that the more components are used, the more information is retained from the original dataset. This allows the Mean Shift algorithm to better identify the clusters in the data. Also the bandwidth used increases with the number of components used in the PCA.

The next plots show the fit time and the predict time related to the PCA dimension for the best scores obtained.

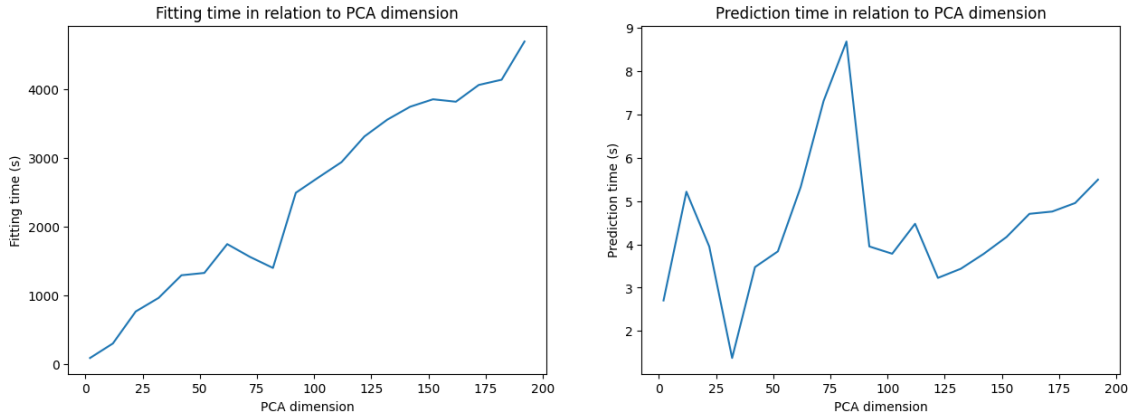

Figure 4.6: Time analysis of the Mean shift with MNIST dataset in relation to the PCA dimension

As we can see from the first plot the fit time increases with the number of components used in the PCA, while the predict time remains almost constant.

The following plot shows the number of clusters obtained from the application of the Mean Shift algorithm on the MNIST dataset compared to the number of components used in the PCA.

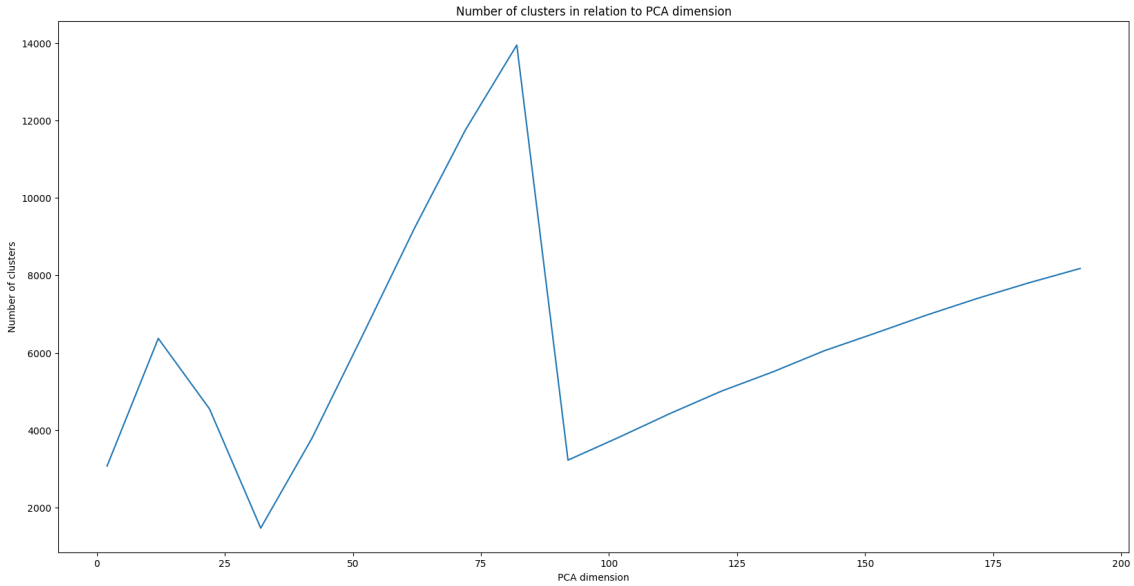

Figure 4.7: Cluster number analysis of the Mean shift with MNIST dataset in relation to the PCA dimension

As we can see from the plot, the number of clusters seems to increase with the number of components used in the PCA, but it's important to notice that in the graph showed the number of clusters drops when the bandwidth value is increased.

The next plot shows the cluster obtained from the application of the Mean Shift algorithm on the MNIST dataset with a PCA dimension of 2.

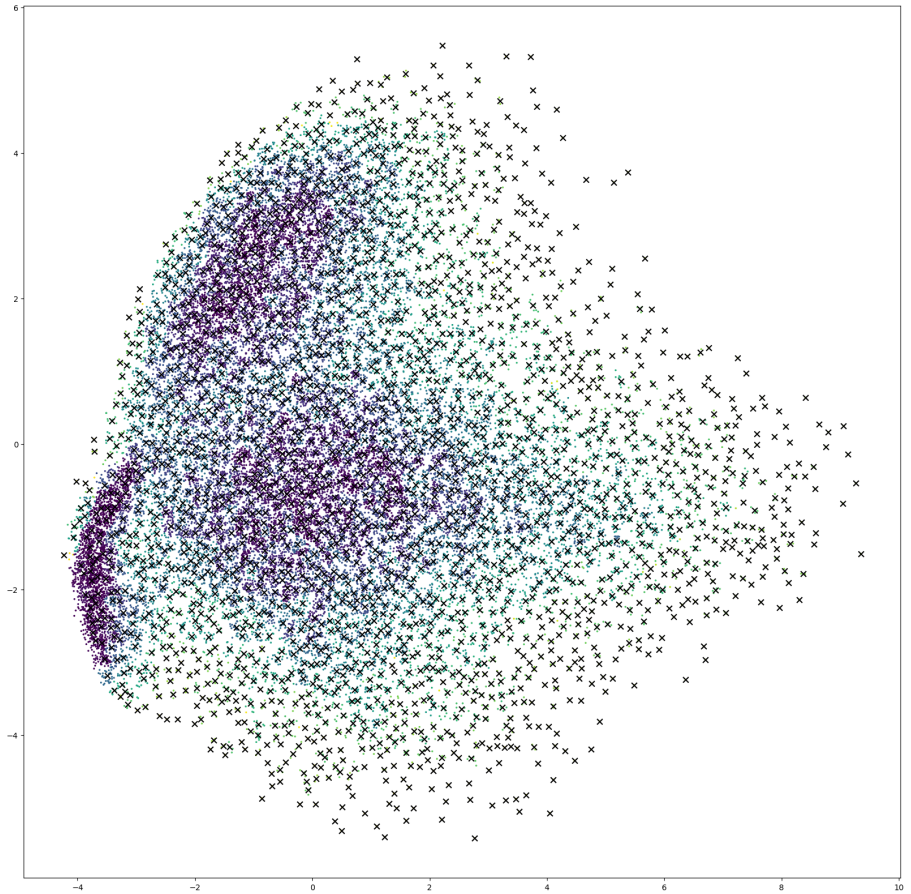

Figure 4.8: Cluster analysis of the Mean shift with MNIST dataset with a PCA dimension of 2

As we can see from the plot, the clusters are not well defined and the algorithm is not able to identify the different digits in the dataset. This is due to the fact that the PCA dimension is too low and the algorithm is not able to retain enough information from the original dataset.

Chapter 5

Normalized Cut

5.1 Defining Normalized Cut

Let $G = (V, E)$ be an undirected weighted graph, where V is the set of vertices and E is the set of edges. Let W be the weight matrix of the graph, where w_{ij} is the weight of the edge between vertices i and j. The normalized cut of a graph is a measure of how well the graph G can be partitioned into two disjoint sets such that the edges between the two sets are minimized and the edges within each set are maximized. The normalized cut of a partition A of the graph is defined as follows:

Definition 5 (Normalized Cut)**.** *The normalized cut of a partition A of the graph is defined as:*

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

where $cut(A, B)$ is the sum of the weights of the edges between the two sets, defined as:

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

and $vol(A) = \sum_{i \in A} d_i$ is the sum of the weights of the edges incident to the vertices in the set, where $d_i = \sum_j w_{ij}$ is the degree of vertex i.

The problem of finding the partition that minimizes the normalized cut is NP-hard. However, the problem can be approximated by solving the eigenvector problem of the graph Laplacian matrix.

5.1.1 Why not minimum cut?

The minimum cut of this graph identifies an optimal partitioning of the data and it is solvable in polynomial time However, the minimum cut is not a good measure of the quality of the partitioning because it does not take into account the size of the partitions. The normalized cut, on the other hand, is a better measure of the quality of the partitioning because it takes into account the size of the partitions. The minimum cut favors highly unbalanced partitions, while the normalized cut favors balanced partitions.

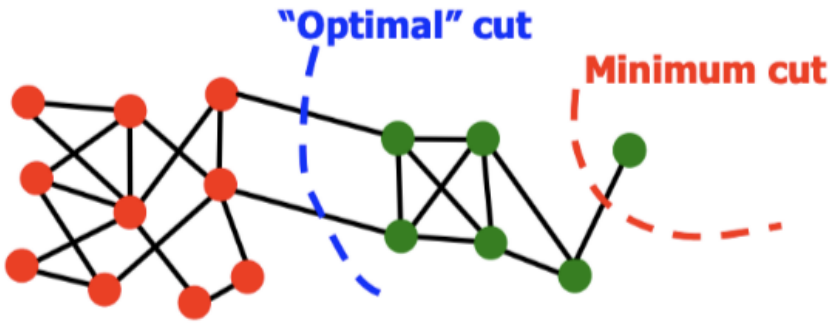

Figure 5.1: Normalized Cut vs Minimum Cut.

5.2 Graph Laplacian

The graph Laplacian matrix of a graph is defined as:

Definition 6 (Graph Laplacian)**.** *The graph Laplacian matrix L of a graph is defined as:*

$$L = D - W$$

where D is the degree matrix of the graph, defined as:

$$D = diag(d_1, d_2, \ldots, d_n)$$

and W is the adjacency matrix of the graph.

The graph Laplacian matrix is a symmetric positive semi-definite matrix implying that all its eigenvalues are non-negative. The smallest eigenvalue of the graph Laplacian matrix is zero, and the corresponding eigenvector is the constant vector.

The graph Laplacian allows us to define the normalized cut in terms of the eigenvectors of the graph Laplacian matrix, giving us an efficient way to approximate the solution to the normalized cut problem.

5.2.1 Solving the normalized cut problem

The problem of finding the partition that minimizes the normalized cut can be approximated by solving the eigenvector problem of the graph Laplacian matrix. Using the Rayleigh quotient, we can show that:

$$\min_x \mathrm{Ncut}(x) = \min_y \frac{y^T L y}{y^T D y}$$

where y is the eigenvector of the graph Laplacian matrix corresponding to the second smallest eigenvalue. The eigenvector y can be computed by solving the generalized eigenvalue problem:

$$Ly = \lambda Dy \text{ subject to } y^T D y = 1$$

where λ is the eigenvalue corresponding to the eigenvector y. So, eigenvalues and eigenvectors of the graph Laplacian matrix can be used to approximate the solution to the normalized cut problem.

5.3 Spectral Clustering

Spectral clustering is a method that uses the idea presented above to partition a graph into two or more clusters. The method involves the following steps:

1. Construct the adjacency matrix of the graph.
2. Compute the graph normalized Laplacian matrix L_{sym} defined as:

$$L_{sym} = D^{-1/2} L D^{-1/2}$$

. This matrix allows us to obtain the eigenvectors of the graph Laplacian matrix that are invariant to the rescaling and shifting of the data.

3. Compute the eigenvectors of the graph Laplacian matrix corresponding to the k smallest eigenvalues, where k is the number of clusters (taken as input).
4. Form a matrix U by stacking the eigenvectors as columns.

$$U = [u_1, u_2, \ldots, u_k]$$

5. Normalize the rows of the matrix U as follows:

$$U_{ij} = \frac{U_{ij}}{\sqrt{\sum_k U_{ik}^2}}$$

. This step is important because ensures that the eigenvectors have the same scale.

6. Apply k-means clustering to the rows of the matrix U to obtain the final clusters.

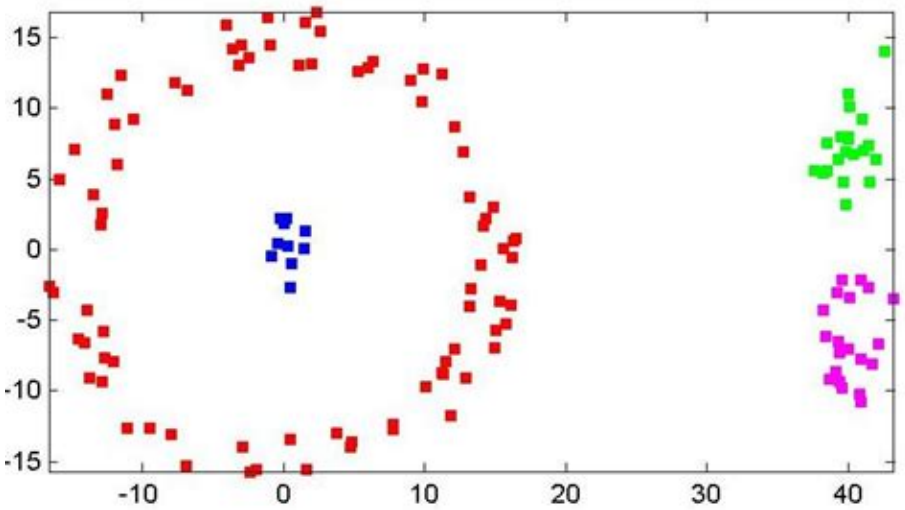

Figure 5.2: Spectral Clustering behavior.

As shown in Figure 5.2, spectral clustering is able to find non-linear decision boundaries that separate the data into clusters. This is because spectral clustering uses the eigenvectors of the graph Laplacian matrix to embed the data into a higher-dimensional space where the clusters are linearly separable. The k-means algorithm is then applied to the embedded data to find the clusters.

5.4 Normalized Cut with MNIST dataset

In this section we will apply the normalized cut algorithm to the MNIST dataset. The library used for the implementation is `scikit-learn` and the parameters of the model are the following:

- `n_clusters`: number of clusters to form.
- `affinity`: the affinity matrix used to compute the graph.
- `assign_labels`: the strategy used to assign labels in the embedding space.

The parameters used for the implementation are the following:

- `n_clusters`: from 5 to 16
- `affinity`: `nearest_neighbors`
- `assign_labels`: `kmeans`

Also, the model has been applied on data obtained from the application of PCA to the MNIST dataset using from 2 to 200 (interleaving 5) components.

5.4.1 Outputs

The following table shows the results obtained from the application of the normalized cut on the MNIST dataset using the parameters described above. The table shows the number of clusters, the rand score and the fit predict time. This is for the best result of each number of components used in the PCA.

Table 5.1: Normalized cut with MNIST dataset results

PCA	n. clusters	score	fit predict time
2	15	0.872653	169.116710
7	15	0.864630	2264.994906
12	10	0.887012	2015.056602
17	15	0.923959	1531.309231
22	14	0.937599	1540.628326
27	14	0.940237	1462.840696
32	13	0.942154	1598.618226
37	13	0.942958	1462.150051
42	13	0.941952	1778.734573
47	13	0.941720	1179.313632
52	13	0.942106	1632.361079
57	13	0.941598	2427.569823
62	13	0.941697	1958.774519
67	13	0.941553	1479.737250
72	13	0.941249	1662.887581
77	13	0.940980	1792.035344
82	13	0.940824	1799.942949
87	13	0.940861	1418.413248
92	13	0.940491	1780.848146
97	13	0.940660	2289.288672

Continued on next page

Table 5.1 – *Continued from previous page*

PCA	n. clusters	score	fit predict time
102	13	0.940638	2030.028211
107	13	0.940545	1642.978597
112	13	0.940435	2079.069646
117	13	0.940241	2351.462018
122	13	0.940343	2170.995635
127	13	0.940040	2233.757144
132	13	0.940240	2315.820688
137	13	0.940194	2234.821695
142	13	0.940220	2548.984206
147	13	0.940027	2344.870908
152	13	0.939943	2393.984951
157	13	0.939637	1665.688245
162	13	0.939731	1987.221335
167	13	0.939915	2108.233684
172	13	0.939418	2469.994684
177	13	0.939561	2395.806705
182	13	0.939318	2394.900509
187	13	0.939410	2000.644422
192	13	0.939430	2218.356111
197	13	0.939217	2029.488109

Overall, the best result was obtained using 37 components in the PCA, with 13 clusters and a rand score of 0.942958. The fit predict time was 1462.150051 seconds. The results show that the normalized cut algorithm is able to find the clusters in the MNIST dataset with a high accuracy. The results also show that the algorithm is able to find the clusters in a reasonable amount of time.

5.4.2 Analysis

The following plots show respectively the best score related to the PCA dimension and the number of clusters used.

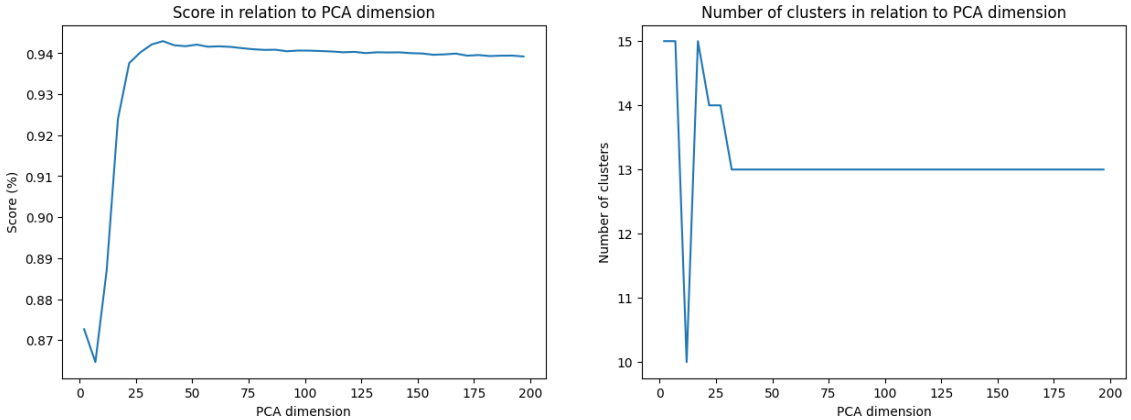

Figure 5.3: Score analysis of the normalized cut with MNIST dataset in relation to the PCA dimension

As shown in Figure 5.3, the score of the normalized cut algorithm increases as the

number of components in the PCA increases up to 37 components. After that, the score remains stable. This suggests that the algorithm is able to find the clusters in the MNIST dataset with a high accuracy using 37 components in the PCA. Also, the number of clusters used for the best results is almost constant around 13.

The next plots show the fit predict time related to the PCA dimension for the best scores obtained.

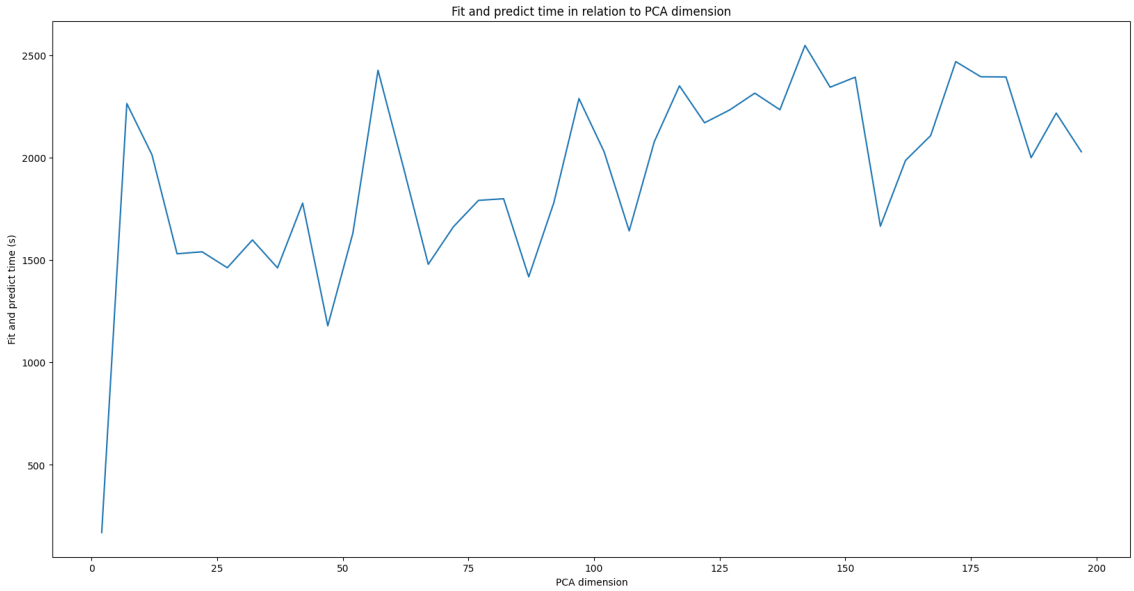

Figure 5.4: Time analysis of the normalized cut with MNIST dataset in relation to the PCA dimension

As shown in Figure 5.4, the fit predict time of the normalized cut algorithm increases as the number of components in the PCA increases initially. After that, the fit predict time remains stable. This suggests that the algorithm is able to find the clusters in the MNIST dataset in a reasonable amount of time. Notice that we cannot distinguish the time related to the fit and the predict phases since the normalize cut algorithm can only cluster data during the fit phase.

The next plot shows the cluster obtained from the application of the normalized cut algorithm on the MNIST dataset with a PCA dimension of 2.

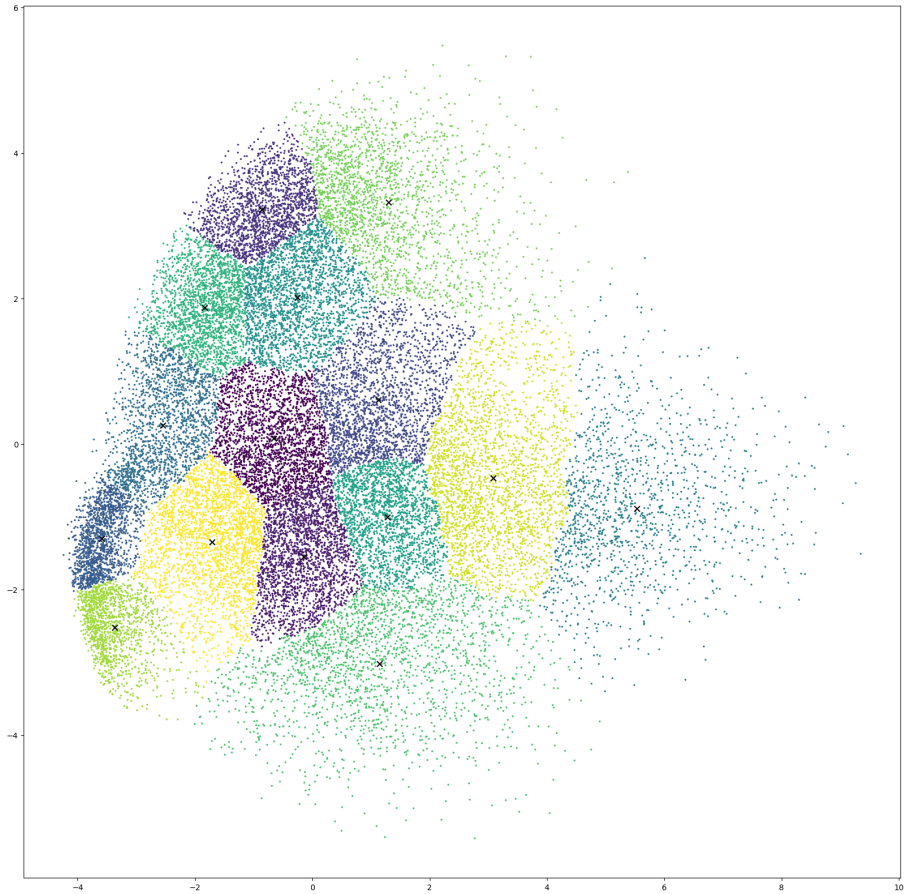

Figure 5.5: Clusters obtained from the application of the normalized cut algorithm on the MNIST dataset with a PCA dimension of 2

Chapter 6

Conclusions

6.1 Models comparison

In this section we will compare the models we have implemented in the previous chapters. We will compare the models in terms of their performance and their rand score. The following table shows the best result obtained by each model.

Model	PCA	Rand Score	Number of clusters
Gaussian Mixture	17	0.8996	15
Mean Shift	42	0.9071	3793
Normalized Cut	37	0.9430	13

Table 6.1: Best results obtained by each model

The best model in terms of Rand Score is the Normalized Cut model, with a score of 0.9430. The Mean Shift model seems to be the second best model, with a Rand Score of 0.9071, However, the number of clusters obtained by this model is 3793, which is not a reasonable number of clusters for the dataset we are working with. The Gaussian Mixture model is the worst model in terms of Rand Score, with a score of 0.8996. However, the number of clusters obtained by this model is 15, which is a reasonable number of clusters.

In the next table we will compare the performance of the models in terms of the time they take to fit and predict.

Model	PCA	Fit time	Predict time	Total time
Gaussian Mixture	17	1.752979	0.032549	1.785528
Mean Shift	42	1291.102779	3.475688	1294.578467
Normalized Cut	37	1462.150051	/	1462.150051

Table 6.2: Performance of the models

The Gaussian Mixture model is the fastest model both in terms of fit time and predict time. On the other hand, the normalized cut model is the slowest model in terms of fit and predict time. Finally, the Mean Shift model requires a bit less time than the Normalized Cut model.

Overall, taking care of the Rand Score and the time required to fit and predict, the best model is the Gaussian Mixture model. This model has a reasonable number of clusters and a good Rand Score. Moreover, it is the fastest model in terms of fit and predict time. Last thing to mention is that the Gaussian Mixture model is also the model that works better with less number of features, as we can see in the table

above, the PCA with 17 components is the best result obtained by this model. This is an important point to take into account, as the number of features is a key factor in the performance of the models and also in the physical space required to store the data.

Bibliografia

[1] Torsello Andrea. *Introduction to Artificial Intelligence*. Slide. Course: Introduction to Artificial Intelligence. 2023 (cit. on pp. 4, 8).

[2] GeeksforGeeks. *Dimensionality Reduction*. Website. URL: `https://www.geeksforgeeks.org/dimensionality-reduction/`. 2023 (cit. on p. 6).

[3] GeeksforGeeks. *ML Mean Shift Clustering*. Website. URL: `https://www.geeksforgeeks.org/ml-mean-shift-clustering/`. 2023 (cit. on p. 23).

[4] Pelillo Marcello. *Introduction to Machine Learning*. Slide. Course: Introduction to Machine Learning. 2023 (cit. on p. 5).

[5] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 4th. New York, NY: Pearson, 2023. ISBN: 978-0134610993 (cit. on pp. 12, 13).

[6] Statology. *Rand Index*. Website. URL: `https://www.statology.org/rand-index/`. 2023 (cit. on p. 5).