CA' FOSCARI UNIVERSITY - VENICE

Department of Enviroment sciences, Informatics and Statistics

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

Handwritten digit clustering for the MNIST database

Student:
Michele Lotto 875922

a.y. 2022-23

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 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 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:

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

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 The MNIST database

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems [1]. It was created by "re-mixing" the samples from NIST's original datasets, from which it takes the name [2]. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels [1].

Chapter 2

Unsupervised Learning

Unsupervised learning is associated with learning without supervision or training [3] [4]. In unsupervised learning, the algorithms are trained with data which is neither labeled nor classified [3] [4]. Unsupervised learning can be classified into three categories: [3] [4]

- Clustering;
- Dimensionality Reduction;
- Association.

For the aim of understanding this project, only the first two categories are fully explained.

2.1 Clustering

Clustering is a data mining technique which groups unlabeled data based on their similarities or differences [3]. Clustering algorithms are used to process raw, unclassified data objects into groups (called clusters) represented by structures or patterns in the information [4]. Objects in the same group are more similar (in some sense) to each other than to those in other groups [5]. The notion of a "cluster" cannot be precisely defined, which is one of the reasons why there are so many clustering algorithms [5]. There is a common denominator: a group of data objects [5]. A "clustering" is essentially a set of such clusters, usually containing all objects in the data set [5]. Typical cluster models include:

- Distribution-based models: clusters are modeled using statistical distributions. "Gaussian Mixture" is a distribution model [5]. For more details see Chapter 4.
- Density-based models: defines clusters as dense regions in the data space. "Mean Shift" is a density model [5]. For more details see Chapter 5.
- Graph-based models: a clique, that is, a subset of nodes in a graph such that every two nodes in the subset are connected by an edge can be considered as a prototypical

form of cluster. "Normalized Cut" is a graph-based model [5]. For more details see Chapter 6.

2.1.1 Rand Index

The Rand Index or Rand measure is a measure of the similarity between two data clusterings [6]. Given a set of n elements $S = \{s_1, \ldots, s_n\}$ and two partitions of S to compare $X = \{x_1, \ldots, x_r\}$, a partition of S into r subsets, and $Y = \{y_1, \ldots, y_m\}$, a partition of S into m subsets, define the following: [6]

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

Intuitively: [6]

- $a + b$ is the number of agreements between X and Y.
- $c + d$ is the number of disagreements between X and Y.
- $a + b + c + d$ is the total number of pairs.

The Rand Index is defined as: [6]

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

It represents the probability that X and Y will agree on a randomly chosen pair [6]. Thus, it has a value between 0 and 1, with 0 indicating that the two data clusterings do not agree on any pair of points and 1 indicating that the data clusterings are exactly the same [6].

2.2 Dimensionality reduction

Dimensionality reduction is a technique used when the number of features, or dimensions, in a given dataset is too high. It reduces the number of data inputs to a manageable size while also preserving the integrity of the dataset as much as possible [3]; having too much data can impact the performance of machine learning algorithms (e.g. overfitting) and it can also make it difficult to visualize datasets [3].

2.2.1 Principal Component Analysis (PCA)

Principal component analysis (PCA) is a type of dimensionality reduction algorithm which is used to reduce redundancies and to compress datasets through feature extraction [7]. This method uses a linear transformation to create a new data representation, yielding a set of principal components [8] [7]. Let the data be a set of D-dimensional vectors $\{x_1, \ldots, x_n\}$; the first principal component is a unit vector μ (i.e., $\mu^T\mu = 1$) that maximizes the variance of the projected data: [8]

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

where: [8]

- S is the covariance matrix of the data:

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

- $\bar{x}$ is the mean of the data:

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

Thus the vector μ corresponds to the leading eigenvector of matrix S. [8] It is possible to define additional principal components incrementally by choose a new direction μ' that maximizes the variance of the projected data among the vectors orthogonal to the directions already considered [8]. In general the k principal components correspond to the k leading eigenvectors [8]. Thus, by extracting k principal components $\{\mu_1, \ldots, \mu_k\}$, the projected data are k-dimensional vectors $\{\tilde{x}_1, \ldots, \tilde{x}_n\}$, defined as: [8]

$$\tilde{x}_i = \begin{pmatrix} x_i^T\mu_1, & \ldots, & x_i^T\mu_k \end{pmatrix}$$

PCA reconstruction and error

It is possible to approximated the initial set of D-dimensional vectors $\{x_1, \ldots, x_n\}$ from the data mean $\bar{x}$ and the k principal components $\{\mu_1, \ldots, \mu_k\}$ [8]. Each data point can be approximated by a linear combination of components: [8]

$$x_i - \bar{x} \approx \sum_{j=1}^{k} \alpha_{i,j}\ \mu_j$$

or equivalently:

$$x_i \approx \bar{x} + \sum_{j=1}^{k} \alpha_{i,j}\ \mu_j$$

where $\alpha_{i,j}$ are obtained by orthogonal projection:

$$\alpha_{i,j} = (x_i - \bar{x})^T \mu_j$$

Therefore, by choosing $k < D$, the dimension of the data is effectively reduced but some information is lost [8]. The distance between x_i and $\tilde{x}_i$ is: [8]

$$x_i - \tilde{x}_i = \sum_{j=k+1}^{D} ((x_i - \bar{x})^T \mu_j)\mu_j$$

Obviously, if $k = D$ then $x_i - \tilde{x}_i = 0$ [8].

Chapter 3

Project Structure

The project uses Python 3.10.6 64 bit along with the packages: "numpy" [9], "pandas" [10], "scikit-learn" [11], "matplotlib" [12], "tqdm" [13] and "seaborn" [14]. Therefore, it is necessary to install them using "`pip3 install package_name`".
The project includes:

- `tuning.py`, a script for tuning the model hyperparameter and select the best PCA dimension. For more details see Section 3.2;
- 3 notebooks, one for each clustering algorithm:
  - `GaussianMixture.ipynb` for the Gaussians Mixture clustering;
  - `Mean_Shift.ipynb` for the Mean Shift clustering;
  - `Normalized_Cut.ipynb` for the Normalized Cut clustering;

  These notebooks contain analysis and plots for the associated model. For more details see Section 3.4;
- `utility.py`, containing utility functions used in each model notebook. For more details see Section 3.3.
- `dataset.ipynb`, containing initial operations on data. For more details see Section 3.1.
- 5 folders:
  - `pca`: this folder contains the PCA objects. For more details see 3.1.
  - `dataset`: this folder contains the PCA reduced datasets. For more details see 3.1.
  - 3 model folders: these folders contain the result files from the execution of `tuning.py`. For more details see Section 3.2.

The execution pipeline for this project is the following:

1. execution of the notebook `dataset.ipynb`.
2. unzip the files in the model folders or execution of `tuning.py`. For more details see Section 3.2.
3. execution of the chosen model notebook.

The project was executed on "13th Gen Intel i5-13600K (20) @ 5.100GHz" CPU and "Ubuntu 22.04.2 LTS x86_64" OS.

3.1 Notebook "dataset.ipynb"

The "`dataset.ipynb`" notebook has the following purposes:

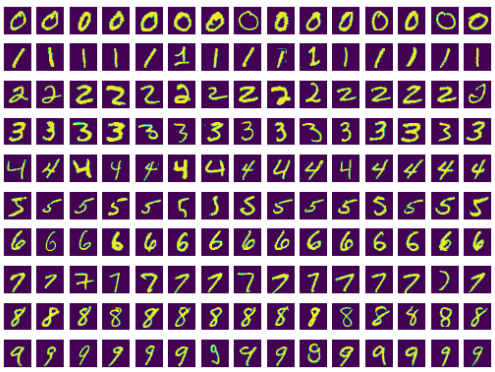
- doing an initial analysis of the data-set;
- creating, for each PCA dimension, a common reduced data-set to use every model;
- saving the reduced data-sets and the initial data-set to file to speed up the computation.

In particular, the following operations are done:

1. fetch database from the net:

```
X,y = fetch_openml('mnist_784', version=1, return_X_y=True)
y = y.astype(int)
X = X/255
```

2. plot some digits from each class:



3. application of the function "`PCA_dfs(200, 0.5)`"; which executes the following steps:
    1. samples of the initial data-set until the target percentage (50%) is reached: the full dataset is too large to be used by all the models; thus, a reasonable small percentage of all the available data was chosen:

```
#random state to ensure replicability
r=np.random.RandomState(32)

#index sampling
indexes=r.choice(70000,int(70000*percentage),replace=False)
```

2. saves the initial data-set to file:

```
X.iloc[indexes].to_parquet("dataset/X.parquet")
y[indexes].to_frame().to_parquet("dataset/y.parquet")
```

3. Creates a PCA reduced data-set from 2 PCA dimensions to 200. For each dimension i:

```
#PCA object with target number of dimensions
pca=PCA(n_components=i)

#PCA reduced dataset for i target dimensions
df=pd.DataFrame(pca.fit_transform(X),
                columns=["PC_"+str(x) for x in range(1,i+1)])

#sample the same indexes as the initial dataset
df=df.iloc[indexes]
```

   Note that the PCA object is fitted with all the initial data and only after the transformation the reduced data-set is sampled with the same indexes of the sampled data-set. This is done to ensure that the PCA transformation is as accurate as possible.

4. saves to file each reduced data-set along with the PCA object to allow PCA reconstruction:

```
#save reduced dataset to file
df.to_parquet("dataset/PCA_"+str(i)+".parquet")

#save PCA object to file
with open("pca/pca_"+str(i)+".pkl", 'wb') as out:
pickle.dump(pca, out, pickle.HIGHEST_PROTOCOL)
```

It is necessary to run `dataset.ipynb` before everything else to ensure the creation of all the data-sets.

3.2 Python file "tuning.py"

The "`tuning.py`" Python script has the purpose of tuning the best hyperparameter for the given model name for each PCA dimension. The script can be simply run by the command:

```
python3 tuning.py model_name
```

where `model_name` is 'GaussianMixture', 'MeanShift' or 'NormalizedCut'.

At the end of the tuning it saves to file the results along with the best fitted models ready to use in a folder named by `model_name`.

The script works as follow:

1. model identification: check if the given keyboard parameter is one of the allowed model names. If it is the case, then it initializes the specified model and the right hyperparameter name along with the hyperparameter values to use to tune the model:

```
match name:
    case "GaussianMixture":
        estimator=GaussianMixture(covariance_type="diag", max_iter=3000,
                                  random_state=32)
        hyperparameter_name="n_components"
        hyperparameter_values=[x for x in range(5,16)]

    case "MeanShift":
        estimator=MeanShift(n_jobs=n_jobs)
        hyperparameter_name="bandwidth"
        hyperparameter_values=[0.2,0.4,0.6,0.8,1,2,5,10,15,20]

    case "NormalizedCut":
        estimator=SpectralClustering(affinity="nearest_neighbors",
                                  neighbors=40, n_jobs=n_jobs)
        hyperparameter_name="n_clusters"
        hyperparameter_values=[x for x in range(5,16)]

    case _:
        print("Wrong model name...")
        exit(1)
```

   The "scikit-learn" implementation of each of the clustering algorithms was used.

   For the "GaussianMixture" clustering, as specified in the assignment instructions, the covariance type is set to be diagonal. The "`max_iter`" parameter is set to 3000 to allow the EM algorithm to converge. For more details see 4.1.1. Additionally, the "`random_state`" parameter is set to 32 to ensure tuning reproducibility.

   For the "NormalizeCut" clustering it was chosen to use the "KNN Graph" to speedup the tuning process. For more details see 6.1.1. Indeed, it was chosen a reasonable small number of neighbors ("`neighbors=40`") on which the KNN Graph is constructed.

   For a full explanation of each model, see Chapter 4 for the Gaussian Mixture model, Chapter 5 for the Mean Shift model and Chapter 6 for the Normalize Cut model.

2. model tuning by the "`get_results( ... )`" function. This function takes in input:

   - "`dfs (Dict[int, pd.DataFrame])`": PCA reduced dataframes, retrieved by the function "`load_PCA_dfs(...)`".

- "`y (pd.Series)`": response dataframe, retrieved by the function "`load_PCA_dfs(...)`".
- "`model (Union[GaussianMixture, MeanShift, SpectralClustering])`": model to tune (defined above).
- "`hyperparameter_name (str)`": hyperparameter to tune (defined above).
- "`hyperparameter_values (List[Union[float, int]])`": list of hyperparameter values (defined above).

The function output is a "`Tuple`" of dictionaries, each of which has a PCA dimension as key argument:

- "`results (Dict[int, pd.DataFrame])`": dict of result dataframes. A result dataframe records the rand index value for each of the hyperparameter values.
- "`best_indexes (Dict[int, int])`": dict of indexes for the best rand index value of the associated result dataframe.
- "
  ```
  fitted_estimators (Dict[int, Union[GaussianMixture,
                                     MeanShift,
                                     SpectralClustering]])
  ```
  ": dict of fitted estimators with the best hyperparameter value for this PCA dimension.
- "`timings (Dict[int, float])`": dict of fitting timings for each PCA dimension.

This function has the purpose of iterating on the available PCA reduced dataframes and calling the auxiliary function "`hyperparameter_tuning(...)`" on each dataframe, which has the porpouse of effectively tuning the hyperparameter for the given PCA reduced dataframe and the given model.

3. save to file: the output of the function "`get_results( ... )`" is saved to file to be used by the associated model notebook.

3.3 Python file: "utility.py"

The `utility.py` Python file has the purpose of making the model notebooks more readable and avoiding code redundancy across them. It includes the following functions:

- "`def load_results( model_name: str ) -> ...`": function that loads from files the results for the specified model. It returns a tuple of dictionaries as the mentioned above "`tuning.get_results( ... )`" function.
- "`def to_frame(results: Dict[int,pd.Series])->pd.DataFrame`": function that given the results dict converts it to pandas dataframe.

- "`def to_latex(results: Dict[int,pd.Series], best_indexes: Dict[int,int])->None`": function that converts the tuning results in latex code.
- "`def load_PCA_dfs() -> Tuple[Dict[int, pd.DataFrame], pd.Series]`": function that loads from file all the PCA reduced datasets along with the ground truth dataset.
- "`def load_df(key:int)->pd.DataFrame`": function that, given a specific PCA dimension, loads the associated PCA reduced dataset.
- "`def best_worst_pca_dimension( ... )->Tuple[int, int, pd.Dataframe]`": function that, given the tuning results and the hyperparameter name:
  - displays the best hyperparameter value for each PCA dimension.
  - finds the best PCA dimension and the worst PCA dimension.
  - plots the effects of PCA dimension over the rand indexes and the hyperparameter values.

  It returns a tuple of best PCA dimension, worst PCA dimension and 'Best n_components for each PCA dimension' pandas dataframe.
- "`def plot_clustering(...)->None`": function that, given the model name and the fitted model with PCA reduced dataset of dimension 2, plots the clustering of PCA dimension 2.
- "`def cluster_composition(...)->None`": function that, given the model name, the fitted models and some PCA dimensions, analyses the content of each cluster of each specified PCA reduced clusterings by creating an heatmap for each of them.
- "`def plot_images_per_cluster(...)->None`": function that, given the model name, the fitted models and some PCA dimensions, plots 4 members of each cluster of each specified PCA reduced clusterings.
- "`def timings_analysis(timings: Dict[int,float], ...)->Tuple[pd.DataFrame,str]`": function that, given the fitting times or the prediction times:
  - plots timings vs PCA dimension;
  - creates a timings dataframe and returns it.
  - generates Latex output and returns it.
- "`def predict_timings(...)->Dict[int,float]`": function that generate prediction times using the given fitted models. Returns a dict of timings, one for each PCA dimension.
- "`def slip_models_file(model_name:str)->None`": function that splits the fitted model file in multiple files.
- "`def merge_models_files(model_name:str, parts:int)->None`": function that merges fitted models files into one file.

3.4 Model notebooks structure

All the model notebooks have a common structure that can be described as follows:

1. initial imports:

```
from utility.py import load_results, to_frame, to_latex
from utility.py import best_worst_pca_dimension, cluster_composition
from utility.py import plot_clustering, plot_images_per_cluster
from utility.py import timings_analysis, predict_timings
```

2. ('NormalizedCut' and 'MeanShift' only) fitting models files merging by the function "`utility.merge_models_files(...)`" to restore the single fitted models file.

3. tuning results loading from file by the function "`utility.load_results(...)`";

4. results displaying as a pandas dataframe by the function "`utility.to_frame(...)`";

5. results displaying with plots with the function "`utility.best_worst_pca_dimension(...)`";

6. 2D PCA plot with the function "`utility.plot_clustering(...)`";

7. clusters composition analysis for the best and the worst PCA dimensions clusterings with the function "`utility.cluster_composition(...)`";

8. plots of sampled cluster member for the best and the worst PCA dimensions clusterings with the function "`utility.plot_images_per_cluster(...)`";

9. (Gaussian Mixture model only) plot of the PCA reconstructed mean for each cluster for the best and the worst PCA dimensions clusterings with the function "`GaussianMixture.plot_means(...)`". For more details see 4.2;

10. fitting times analysis with the function "`utility.timings_analysis(...)`";

11. prediction time analysis with the functions "`utility.timings_analysis(...)`" and "`utility.predict_timings(...)`".

12. ('NormalizedCut' and 'MeanShift' only) fitting models file splitting by the function "`utility.slip_models_file(...)`" to reduce file size.

Chapter 4

Gaussian Mixture

4.1 Theoretical Background

A Gaussian Mixture Model (GMM) is a distribution-based model defined as function comprised of several Gaussians, each identified by $k \in \{1, \ldots, K\}$, where K is the number of clusters in the data: each Gaussian explains the data contained in each of the K clusters available [15] [16]. Each Gaussian k in the mixture is comprised of the following parameters:

- a mean μ_k that defines its centre [15].
- a covariance Σ_k that defines its width or the dimensions of an ellipsoid in a multivariate scenario [15].
- a mixing probability π_k that defines how big or small the Gaussian function will be or in other words the probability that a given observation is explained by Gaussian k [15].

The mixing coefficients creates a probability distribution; thus they must meet this condition: [15] [16]

$$\sum_{k=1}^{K} \pi_k = 1$$

Let z be a binary indicator variable (also called latent variable) defined as: [15] [16]

$$z_k = \begin{cases} 1 & \text{when } x \text{ comes from } \mathcal{N}(x \mid (\mu_k, \Sigma_k)) \\ 0 & \text{otherwise} \end{cases}$$

The overall probability of observing a point that comes from Gaussian k is actually equivalent to the mixing coefficient for that Gaussian: [15] [16]

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

This means that the bigger the Gaussian is, the higher this probability will be [15].

The probability of an observation x, given that it comes from Gaussian k is: [15] [16]

$$\mathbb{P}(x \mid z_k = 1) = \mathcal{N}(x \mid (\mu_k, \Sigma_k))$$

Let Z be the set of all possible variables z_k: [15]

$$Z = \{z_1, \ldots, z_k\}$$

Given that each z_k occurs independently of others and that they can only take the value of one when k is equal to the cluster the point comes from: [15] [16]

$$\begin{aligned}\mathbb{P}(Z) &= \mathbb{P}(z_1 = 1)^{z_1}\mathbb{P}(z_2 = 1)^{z_2}\ldots\mathbb{P}(z_k = 1)^{z_k} \\ &= {\pi_1}^{z_1}{\pi_2}^{z_2}\ldots{\pi_k}^{z_k} \\ &= \prod_{k=1}^{K}{\pi_k}^{z_k}\end{aligned}$$

Following the same logic it is possible to state: [15] [16]

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

These last results can be used to obtain $\mathbb{P}(x)$ by applying a simple marginalization: [15] [16]

$$\begin{aligned}\mathbb{P}(x) &= \sum_{k=1}^{K}\mathbb{P}(x, z_k = 1) \\ &= \sum_{k=1}^{K}\mathbb{P}(x \mid z_k = 1)\mathbb{P}(z_k = 1) \\ &= \sum_{k=1}^{K}\pi_k\,\mathcal{N}(x \mid (\mu_k, \Sigma_k))\end{aligned}$$

This is the equation that defines a Mixture of K Gaussians for each observation [15] [16].

4.1.1 Parameters estimation

Before using a Gaussian Mixture to cluster data it is necessary to estimate the parameters for each Gaussian [15] [16]. Let the parameters of our model be:

$$\theta = \{\pi, \mu, \Sigma\}$$

where:

- $\pi = \{\pi_1, \pi_2, \ldots, \pi_K\}$: the set of all the π_k.
- $\mu = \{\mu_1, \mu_2, \ldots, \mu_k\}$: the set of all the μ_k.

- $\Sigma = \{\Sigma_1, \Sigma_2, \ldots, \Sigma_K\}$: the set of all the Σ_k.

These estimations could be retrieved simply by utilizing Maximum Likelihood Estimation (MLE) [15]:

Let $\mathbb{P}(X)$ be the joint probability of all observations x_i: [15]

$$\mathbb{P}(X) = \prod_{i=1}^{N} \mathbb{P}(x_i) = \prod_{i=1}^{N} \sum_{k=1}^{K} \pi_k \, \mathcal{N}(x_i \mid (\mu_k, \Sigma_k))$$

by applying the log to each side of the equation: [15]

$$\ln \mathbb{P}(X) = \sum_{i=1}^{N} \ln \sum_{k=1}^{K} \pi_k \, \mathcal{N}(x_i \mid (\mu_k, \Sigma_k))$$

Now the next step is differentiate this last equation with respect to each parameter [15]. Unfortunately, calculating the derivative of this expression and then solving for each parameter is a computationally expensive problem, particularly affected by the logarithm of the second summation [15]. The solution is to use an iterative method to estimate the parameters, like the Expectation - Maximization algorithm (EM) [15] [16].

Expectation - Maximization algorithm (EM)

The Expectation - Maximization algorithm is a general technique for finding maximum likelihood estimates for probabilistic models with latent variables [15] [16].

An important result used by EM is the posterior distribution of z_k given an observation: [15] [16]

$$\gamma_i(z_k) = \mathbb{P}(z_k = 1|x_i) = \frac{\mathbb{P}(x_i \mid z_k = 1)\mathbb{P}(z_k = 1)}{\mathbb{P}(x_i)} = \frac{\pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$$

Instead of optimizing $\mathbb{P}(X)$, the EM algorithm works with: [15] [16]

$$\mathbb{P}(X, Z) = \prod_{i=1}^{N} \prod_{k=1}^{K} \pi_k^{z_k} \mathcal{N}(x_i|\mu_k, \Sigma_k)^{z_k}$$

The log-likelihood takes the form: [15]

$$\ln\left(P(X, Z)\right) = \sum_{i=1}^{N} \sum_{k=1}^{K} z_k \left(\ln(\pi_k) + \ln(\mathcal{N}(x_i|\mu_k, \Sigma_k))\right)$$

In this formulation, the logarithm acts directly on the normal density which leads to a simpler solution for the MLE [15].

The expected value of the log-likelihood is therefore: [17]

$$\begin{aligned}
\mathbb{E}_{Z|X}[\ln\left(\mathbb{P}(X, Z)\right)] &= \mathbb{E}_{Z|X}\left[\sum_{i=1}^{N} \sum_{k=1}^{K} z_k \left(\ln(\pi_k) + \ln(\mathcal{N}(x_i \mid \mu_k, \Sigma_k))\right)\right] \\
&= \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbb{E}_{Z|X}[z_k] \left(\ln(\pi_k) + \ln(\mathcal{N}(x_i \mid \mu_k, \Sigma_k))\right)
\end{aligned}$$

Since $\mathbb{E}_{Z|X}[z_k] = \mathbb{P}(z_k = 1 \mid X)$ the expression becomes: [17]

$$\mathbb{E}_{Z|X}[\ln(\mathbb{P}(X, Z))] = \sum_{i=1}^{n} \sum_{k=1}^{K} \gamma_i(z_k) \left(\ln(\pi_k) + \ln(\mathcal{N}(x_i \mid \mu_k, \Sigma_k))\right)$$

This last expression is used by the EM algorithm to estimate θ by maximizing the expected log-likelihood: [15] [17]

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \left(\mathbb{E}_{Z|X}[\ln(\mathbb{P}(X, Z))]\right)$$

Thus, by differentiating the expected log-likelihood with respect to $\theta = (\pi, \mu, \Sigma)$ and by setting the derivatives to zero, we obtain $\hat{\theta} = (\hat{\pi}, \hat{\mu}, \hat{\Sigma})$: [15] [16] [17]

$$\hat{\mu_k} = \frac{\sum_{i=1}^{n} \gamma_i(z_k) x_i}{\sum_{i=1}^{n} \gamma_i(z_k)}$$
$$\hat{\Sigma_k} = \frac{\sum_{i=1}^{n} \gamma_i(z_k)(x_i - \mu_k)^2}{\sum_{i=1}^{n} \gamma_i(z_k)}$$
$$\hat{\pi_k} = \frac{\sum_{i=1}^{n} \gamma_i(z_k)}{N}$$

Follows the EM procedure:

1. **Initialization step**: Initialise θ accordingly [15]. There exist different approaches:

   - use the results obtained by a previous K-Means run [15].
   - running the EM algorithm several times with different, randomly chosen, starting points [15].

   A good initialization is fundamental to avoid using a poor approximation to the true maximum [15].

2. **Expectation step**: Evaluate the posterior probabilities $\gamma_i(z_k)$ using the current θ [15] [17].

3. **Maximization step**: Evaluate the expressions for $\hat{\theta} = (\hat{\pi}, \hat{\mu}, \hat{\Sigma})$ using the previously evaluated $\gamma_i(z_k)$ and μ_k [15] [17].

4. **Evaluation step**: Set $\theta \leftarrow \hat{\theta}$ and evaluate the log-likelihood with the new parameter estimates [15]. If the log-likelihood has changed by less than some small ϵ, stop. Otherwise go back to step 2. [15]

A common optimization for this algorithm is to set a limit of iterations without convergence [15]. This allows the algorithm to stop the computation and return some approximated results even if the convergence is not fully reached [15].

4.1.2 Covariance types

A Gaussian Mixture model can usually use different covariance matrix types:

- "full covariance matrix": each component has its own general covariance matrix.
- "tied covariance": all components share the same general covariance matrix.
- "diagonal covariance matrix": each component has its own diagonal covariance matrix.
- "spherical covariance matrix": each component has its own single variance.

4.2 Using Gaussian Mixture model with MNIST dataset

As mentioned in Chapter 3, the project was executed on "13th Gen Intel i5-13600K (20) @ 5.100GHz" CPU and "Ubuntu 22.04.2 LTS x86_64" OS.

As mentioned in Section 3.2, the tuning parameter for the Gaussian Mixture model is the number of components K (number of clusters).

4.2.1 Results

In table 4.1 the tuning results for each PCA dimension are reported; in each sub-table the best result row is highlighted:

PCA dimension 2	
n_compon.	rand index
5	0.743025
6	0.806998
7	0.822675
8	0.838379
9	0.849437
10	0.855801
11	0.860615
12	0.863414
13	0.868654
14	0.869759
15	0.872862

PCA dimension 12	
n_compon.	rand index
5	0.791927
6	0.812349
7	0.837213
8	0.835781
9	0.846609
10	0.855744
11	0.870195
12	0.887620
13	0.892914
14	0.894916
15	0.888098

PCA dimension 22	
n_compon.	rand index
5	0.780774
6	0.797561
7	0.692609
8	0.825833
9	0.821063
10	0.817118
11	0.853174
12	0.861279
13	0.888257
14	0.892819
15	0.894685

Continued on next page

PCA dimension 32	
n_compon.	rand index
5	0.715862
6	0.727758
7	0.779961
8	0.813561
9	0.856747
10	0.860323
11	0.853495
12	0.858773
13	0.876660
14	0.861830
15	0.886875

PCA dimension 42	
n_compon.	rand index
5	0.719122
6	0.765596
7	0.763399
8	0.818085
9	0.810125
10	0.804219
11	0.827913
12	0.838542
13	0.849081
14	0.848635
15	0.892315

PCA dimension 52	
n_compon.	rand index
5	0.733835
6	0.768919
7	0.768472
8	0.820819
9	0.814654
10	0.810741
11	0.853344
12	0.878500
13	0.867082
14	0.862619
15	0.863059

PCA dimension 62	
n_compon.	rand index
5	0.761449
6	0.775250
7	0.769466
8	0.803109
9	0.796381
10	0.843154
11	0.846613
12	0.857114
13	0.865031
14	0.866768
15	0.850255

PCA dimension 72	
n_compon.	rand index
5	0.761987
6	0.773172
7	0.761712
8	0.782852
9	0.815348
10	0.818622
11	0.826014
12	0.842610
13	0.863372
14	0.837878
15	0.870417

PCA dimension 82	
n_compon.	rand index
5	0.734658
6	0.772638
7	0.790392
8	0.793429
9	0.815299
10	0.816534
11	0.831440
12	0.836191
13	0.830588
14	0.847234
15	0.862029

Continued on next page

PCA dimension 92	
n_compon.	rand index
5	0.712245
6	0.743614
7	0.739819
8	0.777192
9	0.808263
10	0.809899
11	0.817437
12	0.833991
13	0.850245
14	0.855985
15	0.852267

PCA dimension 102	
n_compon.	rand index
5	0.748836
6	0.748050
7	0.747988
8	0.776504
9	0.769370
10	0.812394
11	0.823788
12	0.829534
13	0.839997
14	0.842809
15	0.853669

PCA dimension 112	
n_compon.	rand index
5	0.727882
6	0.743712
7	0.748803
8	0.758165
9	0.773704
10	0.779178
11	0.813420
12	0.799110
13	0.808805
14	0.808059
15	0.818577

PCA dimension 122	
n_compon.	rand index
5	0.715264
6	0.741708
7	0.750288
8	0.738075
9	0.805414
10	0.819948
11	0.792098
12	0.823602
13	0.834333
14	0.839654
15	0.825426

PCA dimension 132	
n_compon.	rand index
5	0.715844
6	0.737845
7	0.765898
8	0.805667
9	0.797312
10	0.792580
11	0.803074
12	0.807333
13	0.821348
14	0.843353
15	0.829681

PCA dimension 142	
n_compon.	rand index
5	0.716369
6	0.735236
7	0.754825
8	0.755696
9	0.801559
10	0.798807
11	0.812090
12	0.816636
13	0.842994
14	0.808337
15	0.825072

Continued on next page

PCA dimension 152	
n_compon.	rand index
5	0.718085
6	0.735625
7	0.759385
8	0.779643
9	0.800278
10	0.762127
11	0.807870
12	0.788385
13	0.803338
14	0.821518
15	0.831916

PCA dimension 162	
n_compon.	rand index
5	0.718731
6	0.733486
7	0.758022
8	0.756687
9	0.800283
10	0.780405
11	0.807015
12	0.800435
13	0.826763
14	0.833553
15	0.839744

PCA dimension 172	
n_compon.	rand index
5	0.718813
6	0.735290
7	0.745019
8	0.798682
9	0.795559
10	0.776009
11	0.821028
12	0.810994
13	0.796432
14	0.801790
15	0.812919

PCA dimension 182	
n_compon.	rand index
5	0.718465
6	0.737931
7	0.746655
8	0.807030
9	0.797718
10	0.794527
11	0.821104
12	0.811555
13	0.794737
14	0.834157
15	0.838471

PCA dimension 192	
n_compon.	rand index
5	0.719936
6	0.739854
7	0.752941
8	0.754484
9	0.800229
10	0.784629
11	0.807092
12	0.823540
13	0.824386
14	0.813783
15	0.834648

Table 4.1: Gaussian Mixture tuning results.

As can be seen from the above tables, the best number of components is always greater than 10 (the true number of classes); in particular in the 75% of cases, the best number of components is 14 or 15. This can be caused by the fact that the higher the number of cluster is, the higher the internal coherence of each cluster is.

In Table 4.2 is reported the best "n_component" value for each PCA dimension:

PCA dim	best n_components	rand_index
2	15.000000	0.872862
12	14.000000	0.894916
22	15.000000	0.894685
32	15.000000	0.886875
42	15.000000	0.892315
52	12.000000	0.878500
62	14.000000	0.866768
72	15.000000	0.870417
82	15.000000	0.862029
92	14.000000	0.855985
102	15.000000	0.853669
112	15.000000	0.818577
122	14.000000	0.839654
132	14.000000	0.843353
142	13.000000	0.842994
152	15.000000	0.831916
162	15.000000	0.839744
172	11.000000	0.821028
182	15.000000	0.838471
192	15.000000	0.834648

Table 4.2: Best n_components for each PCA dimension.

The best PCA dimension is 12 (n_components=14.0) with a rand score of 0.894916, while the worst PCA dimension is 112 (n_components=15.0) with a rand score of 0.818577.

In Figure 4.1 are reported the 'best rand index vs PCA dimension' (LHS) and 'best hyperparameter value vs PCA dimension' (RHS) plots.

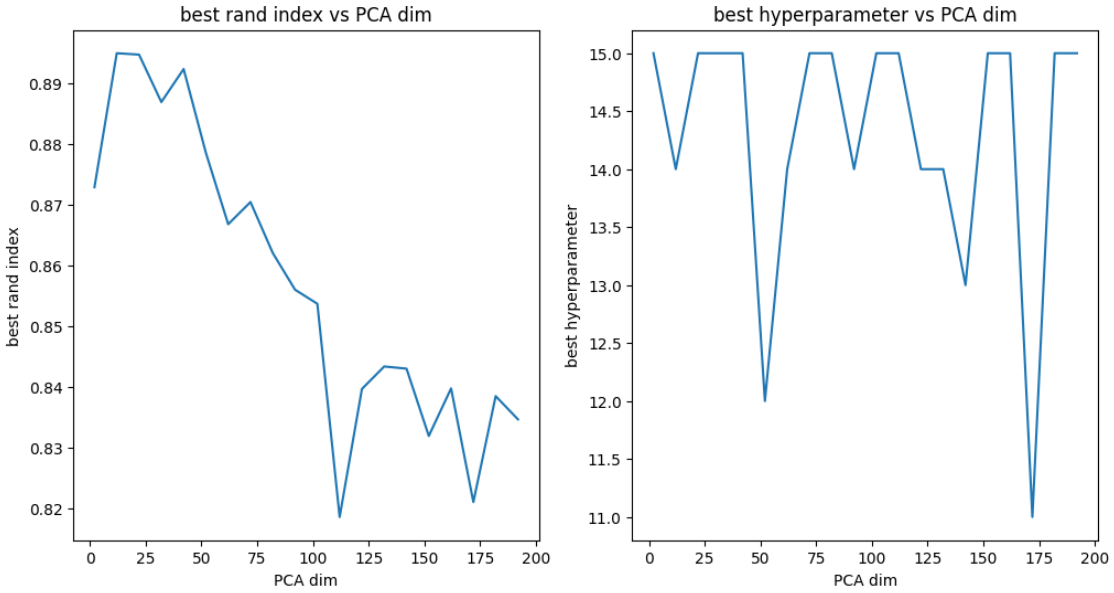


Figure 4.1: 'best rand index vs PCA dimension' (LHS) and 'best hyperparameter value vs PCA dimension' (RHS) plots for the Gaussian Mixture Clustering

As can be seen from the above table and plots:

- the rand index decreases when the PCA dimension increases: the models learn too much noise from the higher PCA dimensions;
- the best hyperparameter value is uncorrelated with the PCA dimension used.

In Figure 4.2 is reported the plot of the best clustering for PCA dimension 2: each color and symbol correspond to a different cluster; the cluster centers are the mean of each component.

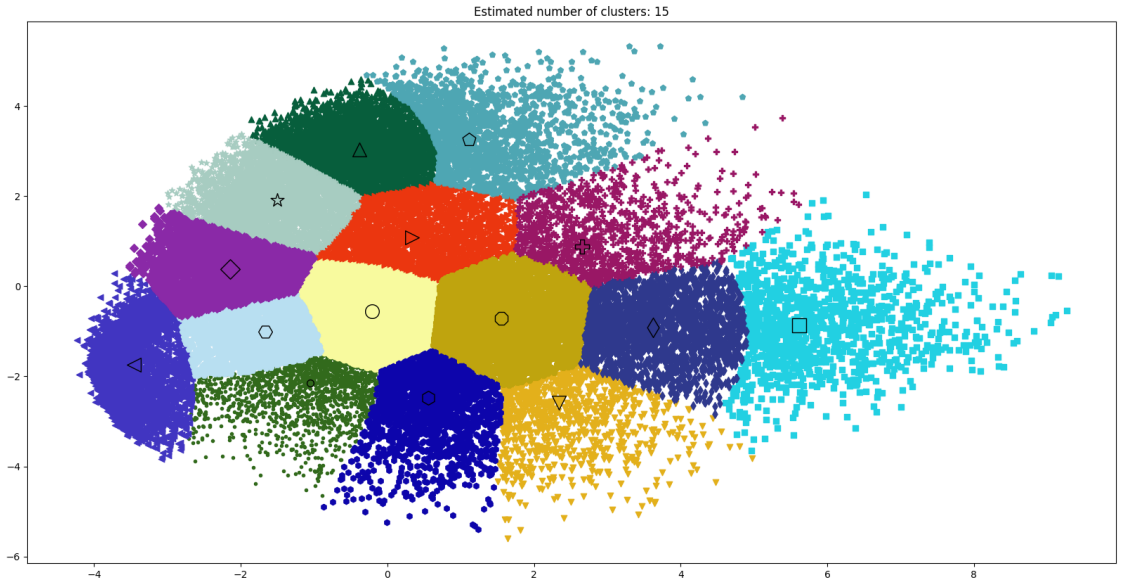


Figure 4.2: Plot of the best clustering for PCA dimension 2.

The PCA reduced data in 2 dimensions appears as an unique blob of points: there are no sharped cluster boundaries.

In Figure 4.3 is reported the cluster composition for each cluster for the best PCA dimension, while in Figure 4.4 for the worst PCA dimension: for each cluster is reported its composition in digit percentages.

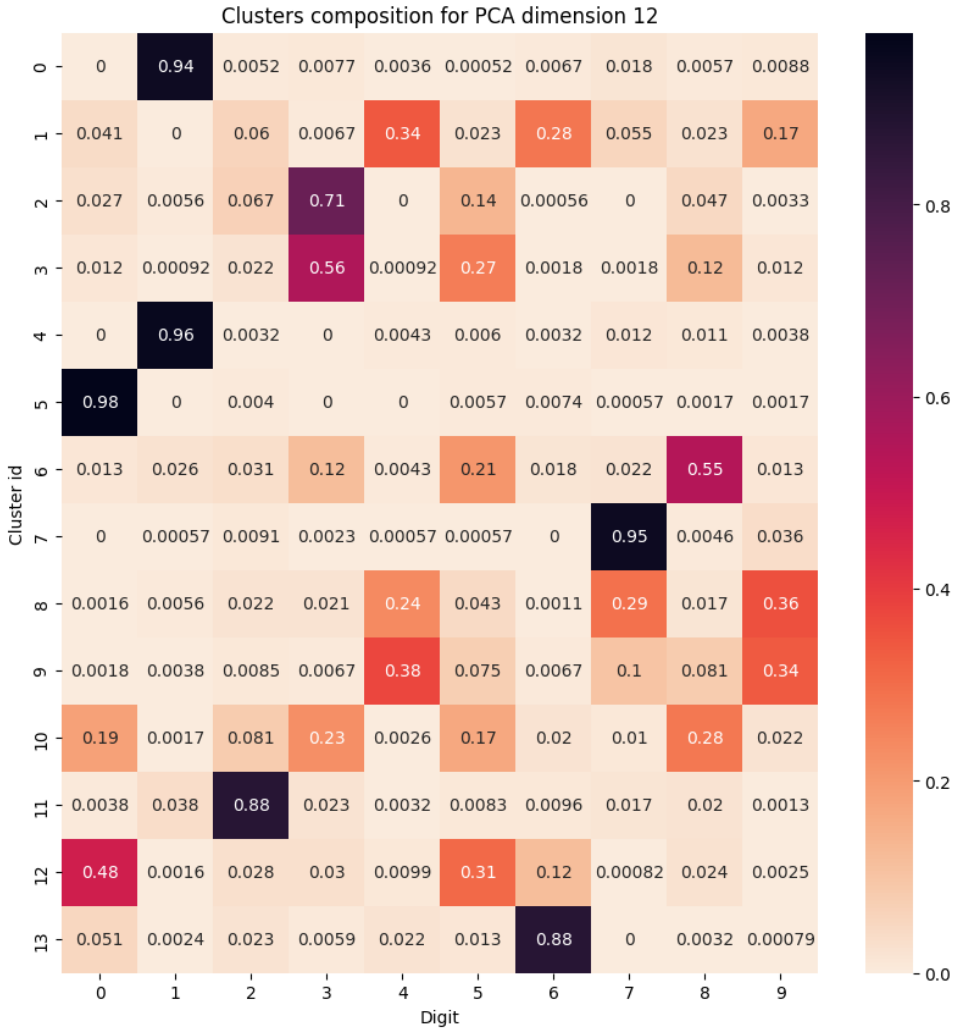


Figure 4.3: Cluster composition for each cluster for the best PCA dimension.

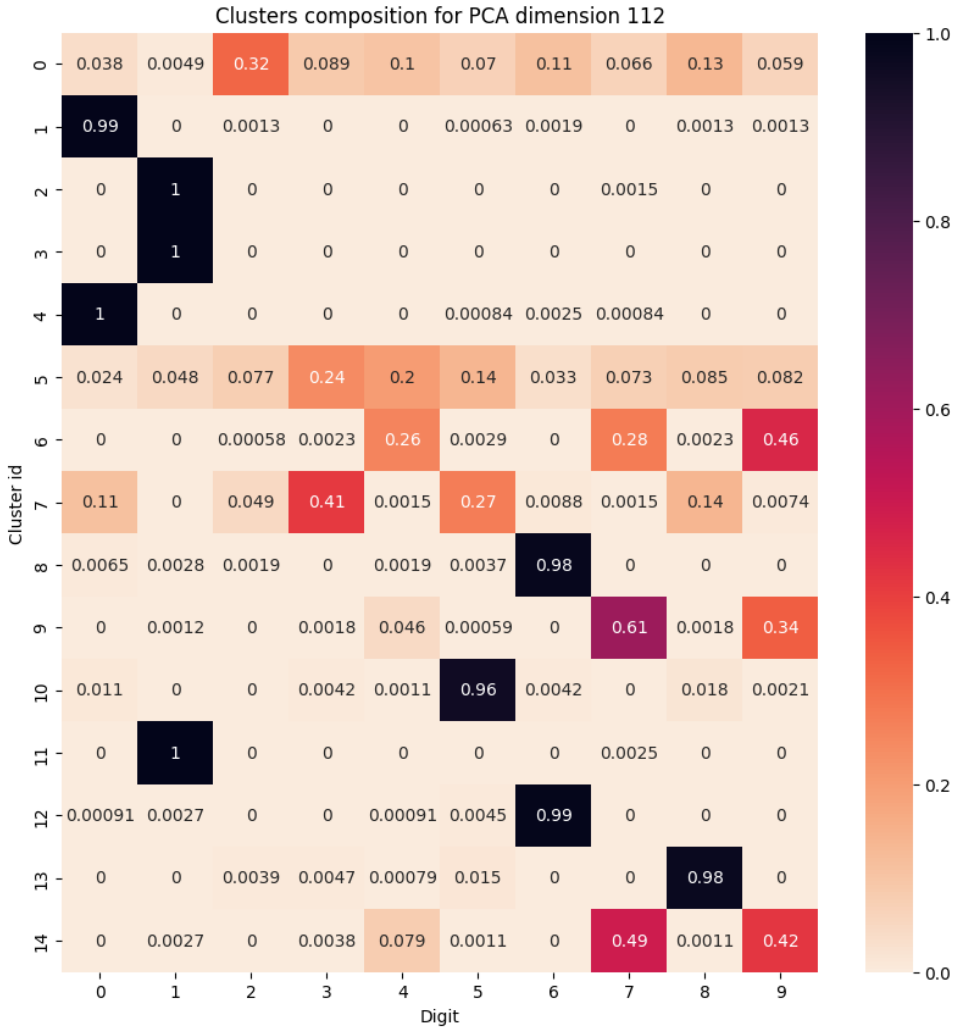


Figure 4.4: Cluster composition for each cluster for the worst PCA dimension.

As can be seen from the above figures, in the best clustering about 43% of clusters have near one digit composition, about 21% of the clusters are above half composed by one digit and about 36% of the clusters are below half or half composed by one digit; while in the worst clustering about 60% of clusters have near one digit composition, only one cluster is above half composed by one digit and about 33% of the clusters are below half or half composed by one digit. The best composed clusters for the best clustering are clusters 0, 4, 5, 7, 11 and 13; while for the worst clustering are clusters 1, 2, 3, 4, 8, 10, 11, 12 and 13. The worst composed clusters for the best clustering are clusters 1, 8, 9, 10 and 12, while for the worst clustering are clusters 0, 5, 6, 7 and 14.

In Figure 4.5 are reported four, randomly chosen, digit images for each cluster for the best PCA dimension clustering and for the worst PCA dimension clustering; the vertical numbers are clusters ids.

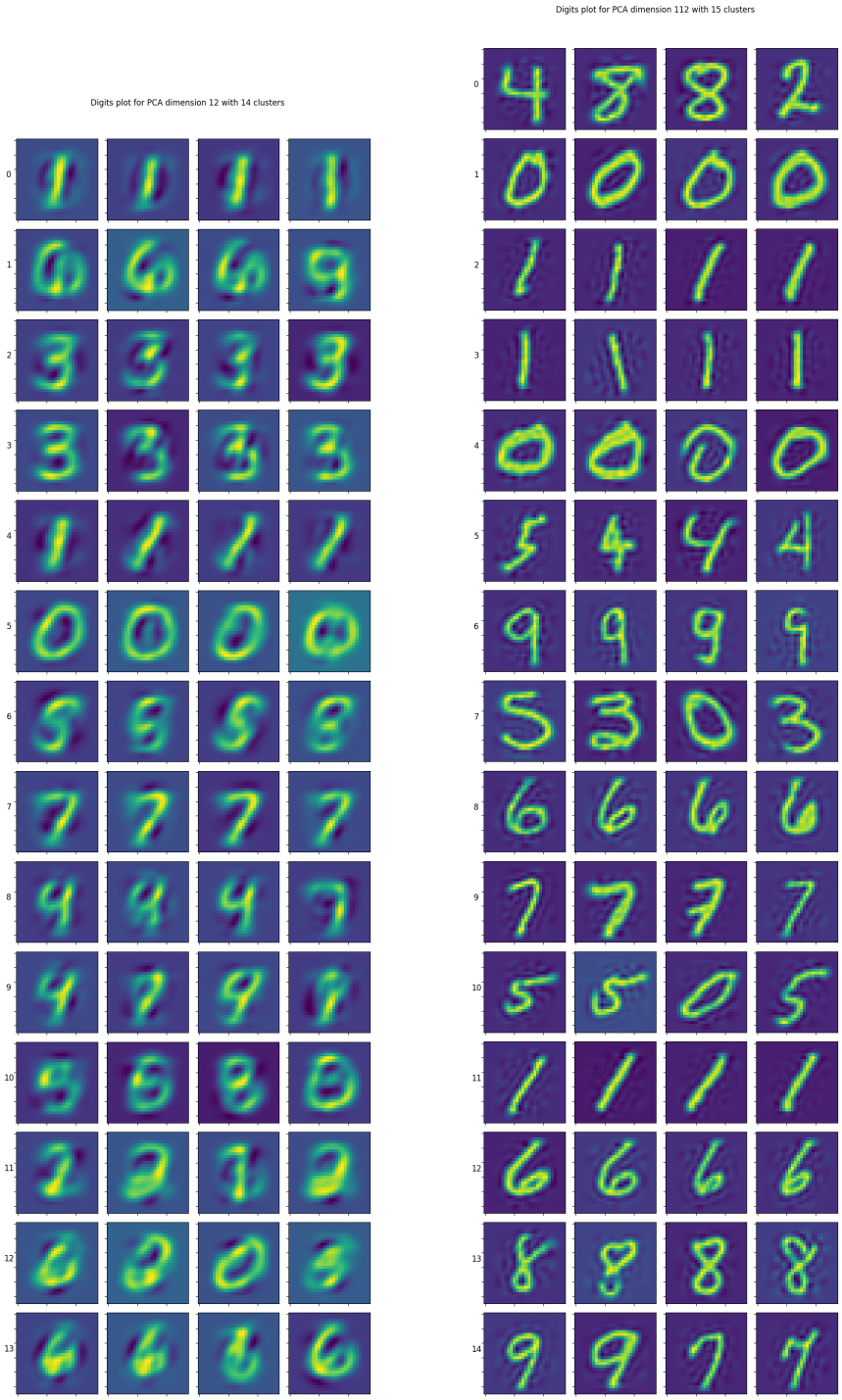


Figure 4.5: Randomly chosen digit images for each cluster for the best PCA dimension clustering (LHS) and for the worst PCA dimension clustering (RHS).

As can be seen in the above image, the best composed clusters for the best clustering contains the digits 1, 0, 7, 2 and 6; while for the worst clustering the digits 0, 1, 6, 5 and 8: this implies that the digits 0, 1 and 6 are the most simple ones to cluster.

As mentioned in Section 1.1, the means of the Gaussian models can be visualized as a greyscale images after PCA reconstruction to inspect the learned model. Thus, in Figure 4.6 are reported the means visualizations for each cluster for the best PCA dimension clustering (LHS) and the worst PCA dimension clustering (RHS).

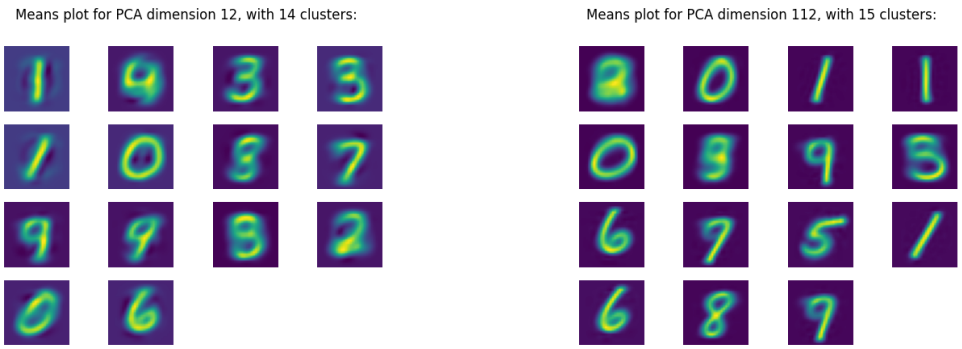


Figure 4.6: Means visualizations for each cluster for the best PCA dimension clustering (LHS) and the worst PCA dimension clustering (RHS).

These images are generated by the function "`plot_means(n_pc:int, means)`" that can be found in the `GaussianMixture` notebook.

4.2.2 Performance

In Table 4.3 are reported the fitting times for the best model for each PCA dimension. In Figure 4.7 these timings are plotted vs the associated PCA dimension.

PCA dim	fitting time
2	0.259092s
12	0.405878s
22	0.617366s
32	0.987784s
42	1.318834s
52	1.253122s
62	1.695571s
72	1.611914s
82	0.905743s
92	0.847506s

Continued on next page

PCA dim	fitting time
102	1.432596s
112	3.213458s
122	1.789425s
132	1.734316s
142	1.942605s
152	3.830632s
162	2.787696s
172	1.784016s
182	3.084397s
192	3.220236s

Table 4.3: Fitting times for the best model for each PCA dimension.

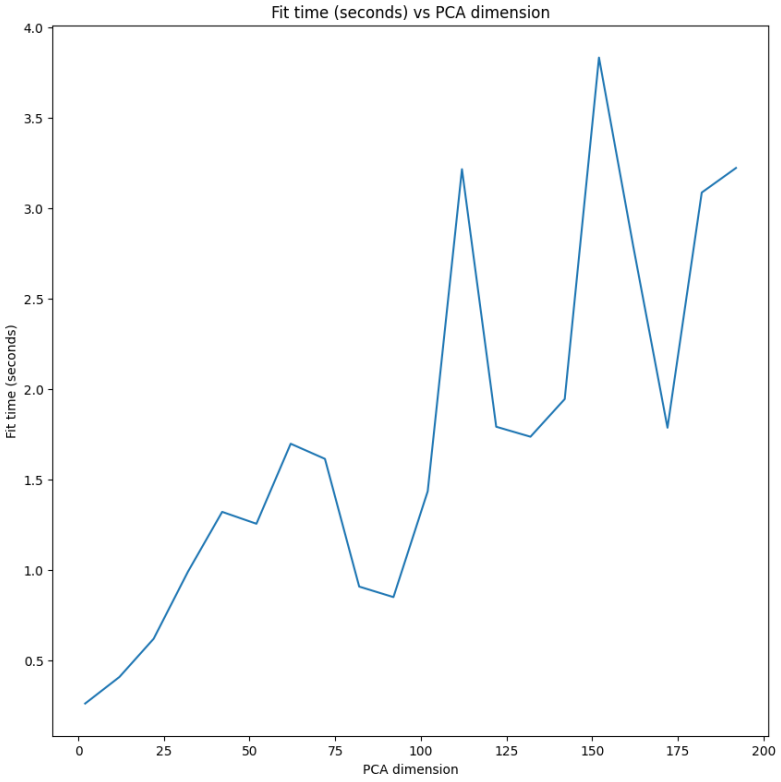


Figure 4.7: Fit time (seconds) vs PCA dimension.

As it is possible to see, the fittings times tend to increase when the PCA dimension increases: more input dimensions implies more computational time. Another observation is that the timings have a large variance; this is caused by the EM algorithm itself: it is sensitive to the parameters initialization. For more details see 4.1.1.

In Table 4.4 are reported the prediction times for the best model for each PCA dimension. In Figure 4.8 these timings are plotted vs the associated PCA dimension.

PCA dim	Prediction time
2	0.004688s
12	0.005108s
22	0.007145s
32	0.005893s
42	0.007145s
52	0.006916s
62	0.008114s
72	0.009392s
82	0.010525s
92	0.011942s
102	0.013095s
112	0.015242s
122	0.019631s
132	0.016493s
142	0.016871s
152	0.017957s
162	0.019125s
172	0.019757s
182	0.021732s
192	0.023228s

Table 4.4: Prediction times for the best model for each PCA dimension.

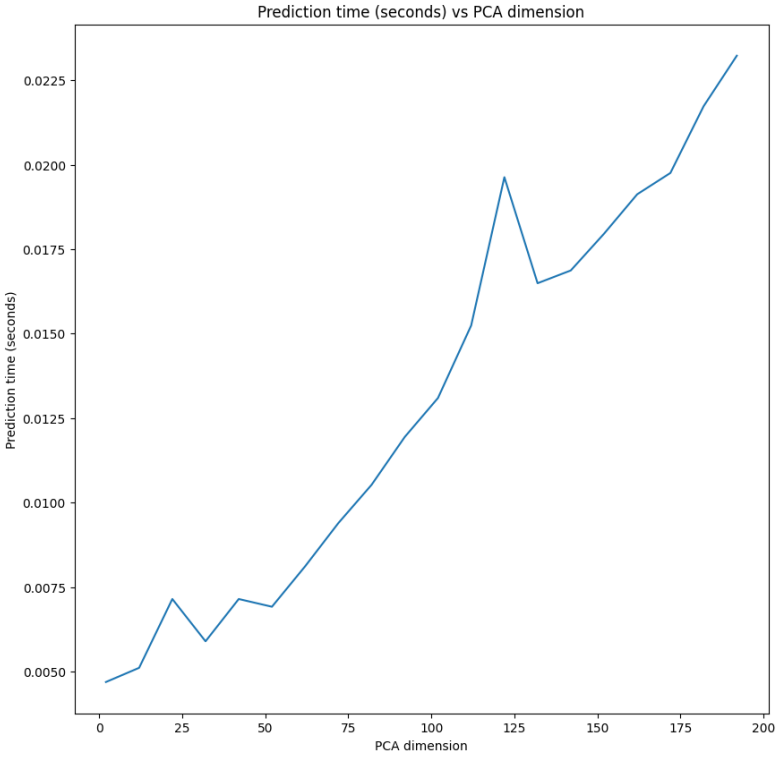


Figure 4.8: Fit time (seconds) vs PCA dimension.

As it is possible to see, the prediction times tends to increase when the PCA dimension increases: more input dimensions implies more computational time. Compared to the fitting times, these timings do not have a large variance.

Chapter 5

Mean Shift

5.1 Theoretical Background

Mean Shift is a density-based clustering model; in particular it is defined as a non-parametric technique for analyzing complex multimodal feature spaces and estimating the stationary points (modes) of the underlying probability density function without explicitly estimating it [16]. The mean shift algorithm seeks the "mode" or point of highest density of a data distribution; each of these points will be the centroid of a cluster [16]. Figure 5.1. This implies that the Mean Shift Algorithm does not require a number of cluster as input.

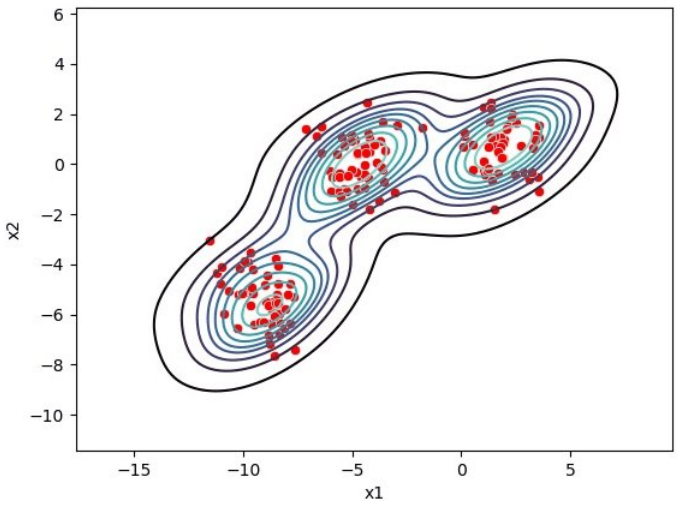


Figure 5.1: Modes visualization [18].

5.1.1 Mean Shift procedure

1. Choose a search window size (h) and the initial location of the search window [16].
2. Compute the mean location ("center of mass" of the data) in the search window [16]. Figure 5.2.

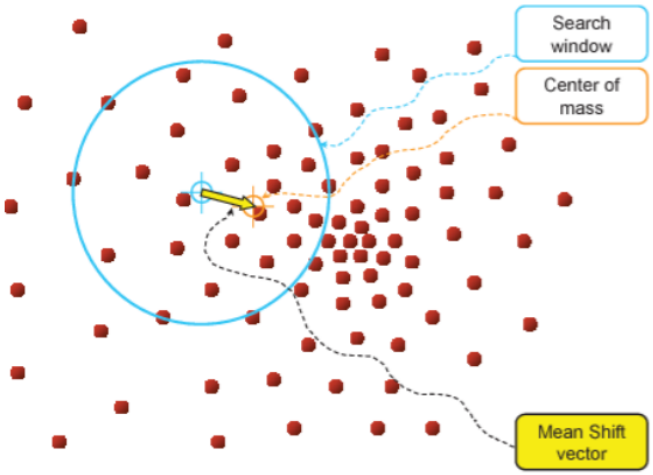


Figure 5.2: Computation of the center of mass for the given search window [16].

3. Center the search window at the mean location computed in Step 2 by "following" the mean shift vector [16]. Figure 5.3.

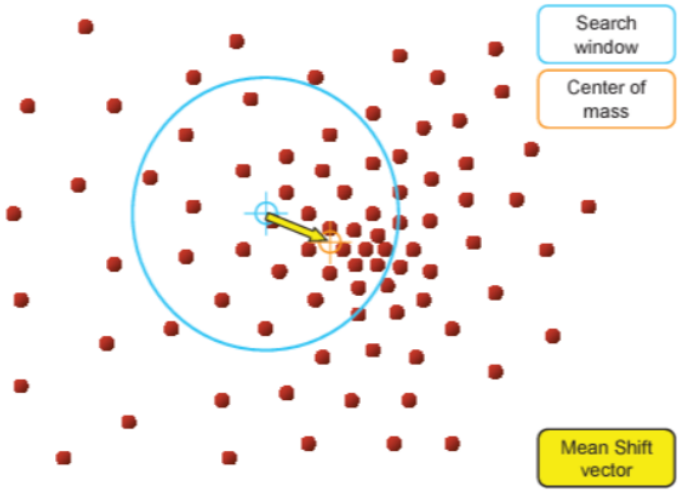


Figure 5.3: Search window centering in the previous center of mass [16].

4. Repeat Steps 2 and 3 until convergence [16]. Figure 5.4.

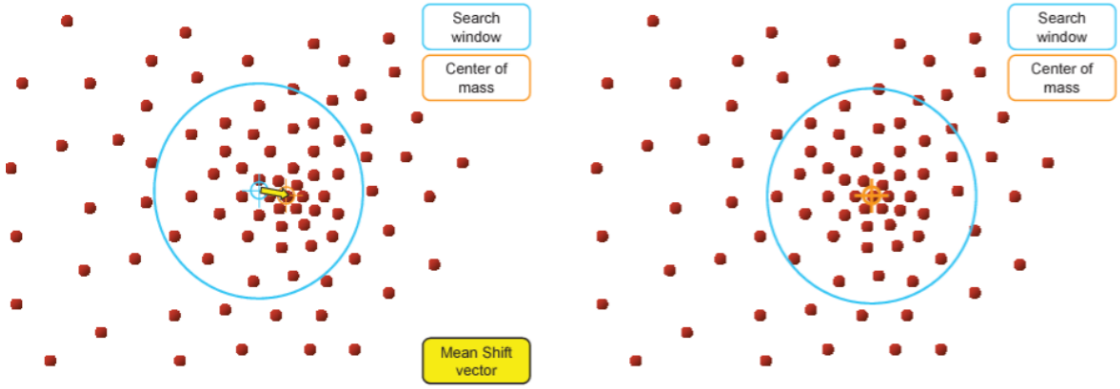


Figure 5.4: Step 2 (left) and step 3 (right) [16].

The position of centroid candidates is iteratively adjusted using a technique called "hill climbing", which finds local maxima of the estimated probability density [19]. Given a candidate centroid x for iteration t, the candidate is updated according to the following equation: [19]

$$x_{t+1} = x_t + m(x_t)$$

where m is the mean shift vector that is computed for each centroid that points towards a region of the maximum increase in the density of points [19]. This vector is derived from the Kernel Density Estimation (5.1.3).

5.1.2 Kernel Density Estimation (Parzen window)

The Kernel Density Estimator (KDE) is an estimation of the unknown density f at any given point x drawn independently from some univariate distribution: [20]

$$\widehat{f}_{h,K}(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\Big(\frac{x - x_i}{h}\Big)$$

where K is the kernel function and $h > 0$ is a smoothing parameter called the bandwidth (size of Parzen window) [20].

Multivariate Kernel Density Estimation

The Kernel Density Estimation can be generalized to the multivariate case by using a multivariate kernel function, usually: [16] [21]

- $K^P(\mathrm{x}) = \prod_{i=1}^{d} \kappa(x_i)$ (product of univariate kernel)

- $K^S(\mathrm{x}) = a_{k,d}\,\kappa(\|x\|)$ (radially symmetric kernel)
- $K(\mathrm{x}) = c_{k,d}\,\kappa(\|x\|^2)$

where:

- d is the number of dimensions;
- $x = (x_1, x_2, \ldots, x_d)^T$ is a d-vector;
- $a_{k,d}$ and $c_{k,d}$ are constants needed by the kernel to satisfy the above properties.

Moreover, the kernel function must: [16] [21]

- be bounded and have compact support.
- be normalized: $\int_{-\infty}^{\infty} K(x)\mathrm{d}x = 1$
- be symmetric: $K(-u) = K(u)$ for all values of u.
- have exponential decay: $\lim_{\|x\|\to\infty} \|x\|^d K(x) = 0$
- be uncorrelated.

Common kernel choices are Epanechnikov, Normal or Uniform [16] [21].

In the multivariate case the bandwidth parameter becomes a symmetric and positive definite $d \times d$ matrix H of univariate bandwidth parameters (one for each dimension) [22]. There are different possible choices of this matrix, usually:

- $\mathrm{H} = h^2\mathrm{I}_d$ (equal bandwidth for each dimension) [16] [22].
- $\mathrm{H} = \mathrm{diag}(h_1^2, h_2^2, \ldots, h_d^2)$ (different bandwidth for different dimensions)[22] .

Thus by using $K(\mathrm{x}) = c_{k,d}\,\kappa(\|x\|^2)$ and $\mathrm{H} = h^2\mathrm{I}_d$, the KDE formula becomes: [22]

$$\widehat{f}_{h,K}(x) = \frac{c_{k,d}}{nh^d}\sum_{i=1}^{n}\kappa\Big(\Big\|\frac{x - x_i}{h}\Big\|^2\Big)$$

where $x = (x_1, x_2, \ldots, x_d)^T$ and $x_i = (x_{i1}, x_{i2}, \ldots, x_{id})^T$ are d-vectors [22].

5.1.3 Computing the Mean Shift vector

As explained in Subsection 5.1.2, KDE estimates the entire probability density function: [20] [16]

$$\widehat{f}_{h,K}(x) = \frac{c_{k,d}}{nh^d}\sum_{i=1}^{n}\kappa\Big(\Big\|\frac{x - x_i}{h}\Big\|^2\Big)$$

On the other hand, the goal of the Mean Shift procedure is to estimate the stationary points (modes) of the underlying probability density function without explicitly estimating it [16] [21]. Thus, to accomplish that, we estimate only the gradient of the PDF: [16] [21]

$$\begin{aligned}\nabla \widehat{f}_{h,K}(x) &= \frac{c_{k,d}}{nh^d}\sum_{i=1}^{n}\nabla\kappa\Big(\Big\|\frac{x-x_i}{h}\Big\|^2\Big) \\ &= \frac{2c_{k,d}}{nh^d}\sum_{i=1}^{n}(x-x_i)g\Big(\Big\|\frac{x-x_i}{h}\Big\|^2\Big) \quad \text{given that } g(x) = -\kappa'(x) \\ &= \frac{2c_{k,d}}{nh^d}\Big[\sum_{i=1}^{n}g\Big(\Big\|\frac{x-x_i}{h}\Big\|^2\Big)\Big]\left[\frac{\sum_{i=1}^{n}x_i\, g\Big(\Big\|\frac{x-x_i}{h}\Big\|^2\Big)}{\sum_{i=1}^{n}g\Big(\Big\|\frac{x-x_i}{h}\Big\|^2\Big)} - x\right]\end{aligned}$$

where:

- $\Big[\sum_{i=1}^{n}g\Big(\Big\|\frac{x-x_i}{h}\Big\|^2\Big)\Big]$ is another Kernel Density Estimation (KDE) [16] [21];
- $\left[\frac{\sum_{i=1}^{n}x_i\, g\Big(\Big\|\frac{x-x_i}{h}\Big\|^2\Big)}{\sum_{i=1}^{n}g\Big(\Big\|\frac{x-x_i}{h}\Big\|^2\Big)} - x\right]$ is the Mean Shift Vector $m(x)$ [16] [21].

Thus, the Mean Shift procedure can be simply described as: [16] [21]

1. compute mean shift vector $m(x)$
2. translate the Kernel window by $m(x)$
3. iterate until convergence

Convergence of gradient ascent algorithms is guaranteed for infinitesimal steps only but the normalization of the Mean Shift vector ensures that it converges [16].

5.1.4 Mean Shift Mode Detection and Clustering

The basin of attraction of a mode is defined as the set of all locations that converges to that mode [16]. Let $\{y_j\}_{j=1,2...}$ denote the sequence of kernel locations [16]. Once y_j gets sufficiently close to a mode of the $\widehat{f}_{h,k}$ it will converge to the mode [16]. At convergence $m(y_c) = 0$ [16].

The Mean Shift procedure is run in parallel to detect multiple modes [16] [21]. The initial locations of the search windows must cover the entire feature space [16] [21]. Figure 5.5.

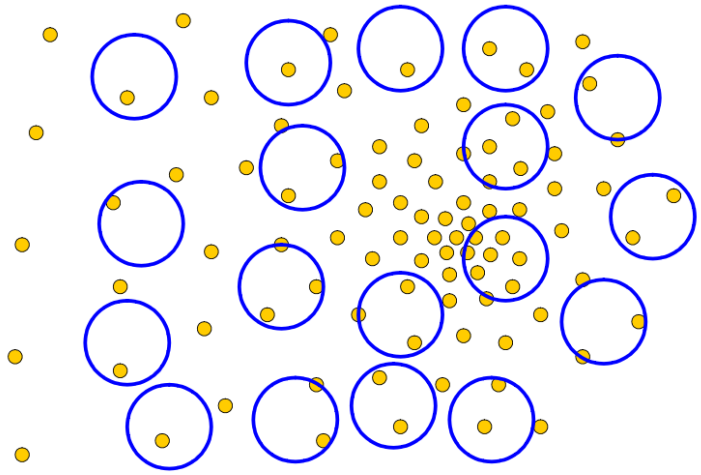


Figure 5.5: Search windows tessellated space [21].

The modes found are merged if they are at a distance less than the bandwidth [16] [21].

Clustering is retrieved from the modes: the basin of attraction of each mode delineates a cluster of arbitrary shape [16] [21]. Figure 5.6.

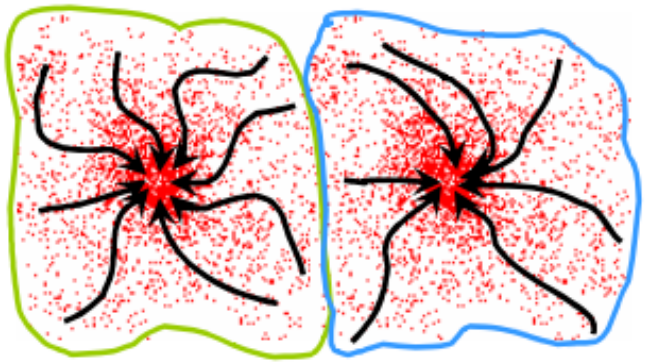


Figure 5.6: Mean Shift Clustering [21].

5.2 Using Mean Shift with MNIST dataset

As mentioned in Chapter 3, the project was executed on "13th Gen Intel i5-13600K (20) @ 5.100GHz" CPU and "Ubuntu 22.04.2 LTS x86_64" OS.

As mentioned in Section 3.2, the tuning parameter for the Mean Shift algorithm is the search window size h (bandwidth).

5.2.1 Results

In table 5.1 the tuning results for each PCA dimension are reported; in each sub-table the best result row is highlighted:

PCA dimension 2		
bandwidth	n_clusters	rand index
0.20	843	0.899584
0.40	118	0.895102
0.60	13	0.846084
0.80	5	0.728909
1.00	5	0.726155
2.00	2	0.524956
5.00	1	0.100226
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 12		
bandwidth	n_clusters	rand index
0.20	34995	0.899774
0.40	34388	0.899776
0.60	32612	0.899800
0.80	31472	0.899913
1.00	29862	0.900306
2.00	5615	0.904387
5.00	1	0.100226
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 22		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	34968	0.899774
0.60	34276	0.899777
0.80	32877	0.899797
1.00	31937	0.899868
2.00	22155	0.902011
5.00	9	0.406508
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 32		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	34996	0.899774
0.60	34816	0.899774
0.80	33799	0.899781
1.00	32662	0.899809
2.00	27733	0.901609
5.00	84	0.710239
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

Continued on next page

PCA dimension 42		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	34925	0.899774
0.80	34315	0.899777
1.00	33172	0.899792
2.00	29562	0.901008
5.00	320	0.852817
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 52		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	34964	0.899774
0.80	34604	0.899775
1.00	33608	0.899784
2.00	30415	0.900622
5.00	715	0.883419
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 62		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	34985	0.899774
0.80	34762	0.899774
1.00	33932	0.899780
2.00	30784	0.900364
5.00	1235	0.897573
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 72		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	34997	0.899774
0.80	34863	0.899774
1.00	34189	0.899778
2.00	31042	0.900273
5.00	1794	0.901562
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 82		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	34999	0.899774
0.80	34909	0.899774
1.00	34364	0.899777
2.00	31205	0.900202
5.00	2354	0.903163
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 92		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	35000	0.899774
0.80	34942	0.899774
1.00	34496	0.899776
2.00	31354	0.900200
5.00	2947	0.905651
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

Continued on next page

PCA dimension 102		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	35000	0.899774
0.80	34954	0.899774
1.00	34602	0.899775
2.00	31444	0.900171
5.00	3430	0.905939
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 112		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	35000	0.899774
0.80	34972	0.899774
1.00	34690	0.899775
2.00	31546	0.900080
5.00	3984	0.906624
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 122		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	35000	0.899774
0.80	34979	0.899774
1.00	34758	0.899775
2.00	31606	0.900075
5.00	4412	0.906457
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 132		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	35000	0.899774
0.80	34983	0.899774
1.00	34819	0.899774
2.00	31662	0.900057
5.00	4843	0.906956
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 142		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	35000	0.899774
0.80	34989	0.899774
1.00	34852	0.899774
2.00	31722	0.900028
5.00	5255	0.906858
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 152		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	35000	0.899774
0.80	34993	0.899774
1.00	34877	0.899774
2.00	31774	0.899988
5.00	5617	0.907319
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

Continued on next page

PCA dimension 162		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	35000	0.899774
0.80	34996	0.899774
1.00	34900	0.899774
2.00	31806	0.899991
5.00	5968	0.906911
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 172		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	35000	0.899774
0.80	34997	0.899774
1.00	34922	0.899774
2.00	31847	0.899978
5.00	6267	0.906778
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 182		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	35000	0.899774
0.80	34999	0.899774
1.00	34938	0.899774
2.00	31890	0.899982
5.00	6592	0.906759
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

PCA dimension 192		
bandwidth	n_clusters	rand index
0.20	35000	0.899774
0.40	35000	0.899774
0.60	35000	0.899774
0.80	35000	0.899774
1.00	34954	0.899774
2.00	31944	0.899948
5.00	6833	0.906875
10.00	1	0.100226
15.00	1	0.100226
20.00	1	0.100226

Table 5.1: Mean Shift tuning results.

As can be seen from the above tables, the number of clusters found is much higher than 10 (the true number of classes). This can be caused by the fact that the higher the number of cluster is, the higher the internal coherence of each cluster is and by the fact that the Mean Shift algorithm has no limit on the number of clusters to find. Another observation is that the best bandwidth is almost always 5 or 2: this means that the PCA dimension has little impact on the search windows size.

In Table 5.2 is reported the best "bandwidth" value for each PCA dimension along with the number of clusters found:

PCA dim	best bandwidth	n_clusters	rand_index
2	0.20	843.00	0.899584
12	2.00	5615.00	0.904387
22	2.00	22155.00	0.902011
32	2.00	27733.00	0.901609
42	2.00	29562.00	0.901008
52	2.00	30415.00	0.900622
62	2.00	30784.00	0.900364
72	5.00	1794.00	0.901562
82	5.00	2354.00	0.903163
92	5.00	2947.00	0.905651
102	5.00	3430.00	0.905939
112	5.00	3984.00	0.906624
122	5.00	4412.00	0.906457
132	5.00	4843.00	0.906956
142	5.00	5255.00	0.906858
152	5.00	5617.00	0.907319
162	5.00	5968.00	0.906911
172	5.00	6267.00	0.906778
182	5.00	6592.00	0.906759
192	5.00	6833.00	0.906875

Table 5.2: Best bandwidth for each PCA dimension.

The best PCA dimension is 152 (bandwidth=5.0, n_clusters=5617) with a rand score of 0.907319, while the worst PCA dimension is 2 (bandwidth=0.2, n_clusters=843) with a rand score of 0.899584.

In Figure 5.7 are reported the 'best rand index vs PCA dimension' (LHS) and the 'best hyperparameter value vs PCA dimension' (RHS) plots.

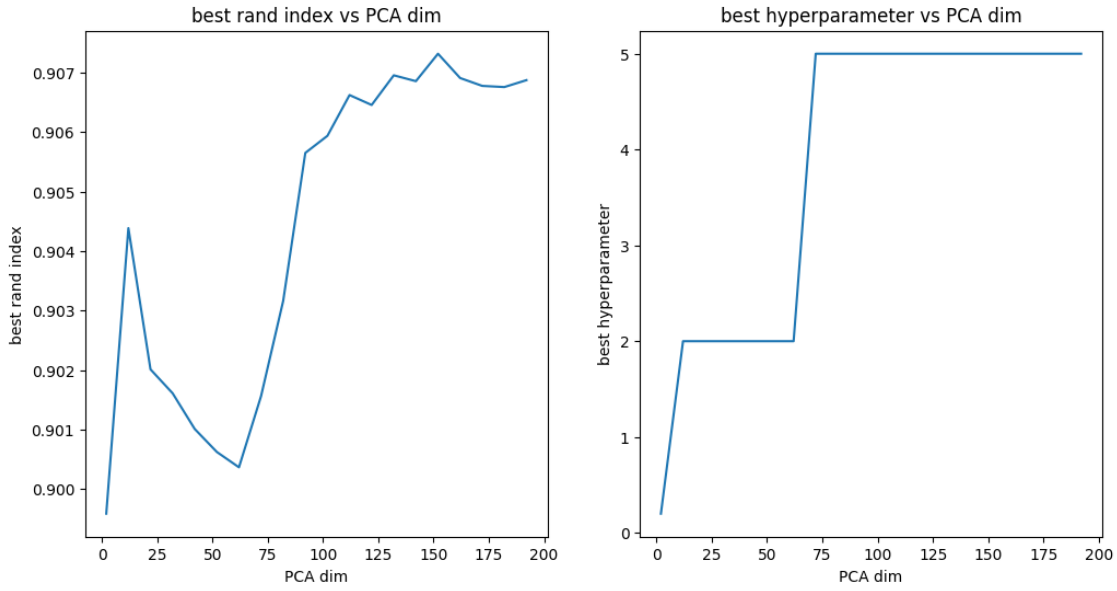


Figure 5.7: 'best rand index vs PCA dimension' (LHS) and 'best hyperparameter value vs PCA dimension' (RHS) plots for the Mean Shift Clustering.

In Figure 5.7 is reported the 'best n_clusters vs PCA dimension' plot.

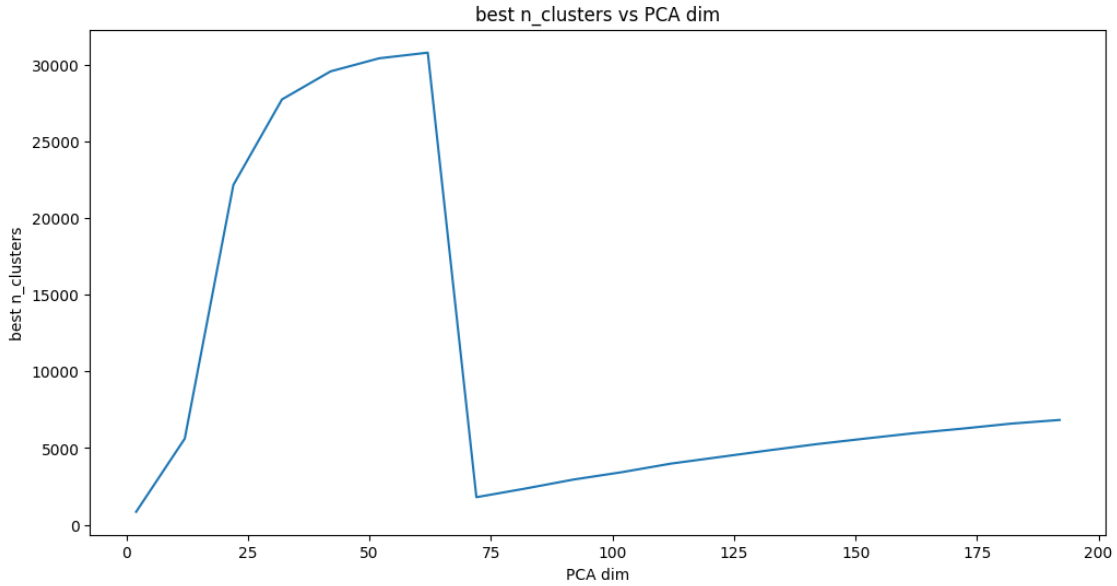


Figure 5.8: 'best n_clusters vs PCA dimension' plot for the Mean Shift Clustering.

As it is possible to see from the above table and plots:

- the rand_index tends to increase when the PCA dimension increases: having no boundaries on the number of clusters allows the algorithm to learn more specific characteristics of the digits by putting different characteristics in different clusters.
- the bandwidth parameter tends to increase when the PCA dimension increase: the more specific characteristics of digits are learned the more it is easy to fall in a local maxima, thus a large bandwidth is preferable.
- the number of clusters tends to increase when the PCA dimension increases; furthermore, when PCA dimension 72 is reached, the number of clusters drops significantly: the drop corresponds to the change in the bandwidth parameter (from 2 to 5).

In Figure 5.9 is reported the plot of the best clustering for PCA dimension 2: each color and symbol correspond to a different cluster; the cluster centers are the mean of each component.

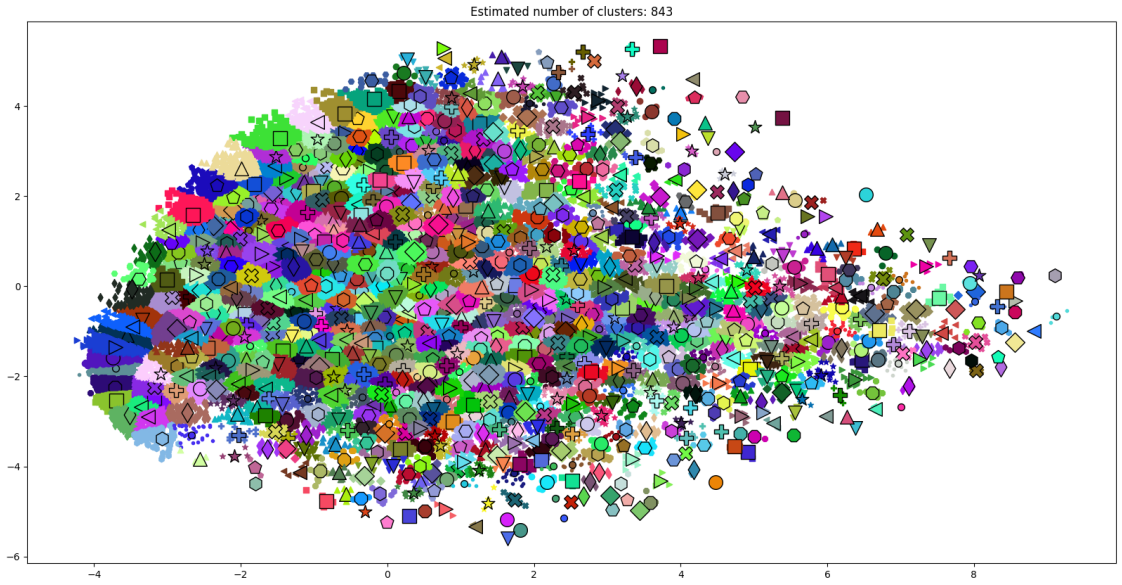


Figure 5.9: Plot of the best clustering for PCA dimension 2.

The PCA reduced data in 2 dimensions appears as an unique blob of points: there are no sharped cluster boundaries.

Due to the high number of clusters in the Mean Shift clustering, the heatmap representation of clusters composition for the best clustering and worst clustering is not possible. The cluster composition is present in the `MeanShift` notebook as a pandas dataframe. Follows some statistics on the each clustering composition:

- For the best clustering (PCA dimension 152):

  - 88.160940% of clusters are near one digit composed;
  - 7.637529% of clusters are above half composed by one digit;
  - 4.201531% of clusters are below half or half composed by one digit.

- For the worst clustering (PCA dimension 2):
  - 24.317912% of clusters are near one digit composed;
  - 25.978648% of clusters are above half composed by one digit;
  - 49.703440% of clusters are below half or half composed by one digit.

As expected, in the best clustering almost all the clusters are one digit composed, while in the worst clustering half of the clusters are below half composed by one digit.

Due to the high number of clusters, it is impossible to show some images for each of them.

5.2.2 Performance

In Table 5.3 are reported the fitting times for the best model for each PCA dimension. In Figure 5.10 these timings are plotted vs the associated PCA dimension.

PCA dim	Fitting time
2	17.431271s
12	81.265699s
22	19.417945s
32	25.854105s
42	25.407525s
52	28.124219s
62	30.327785s
72	311.89021s
82	344.162693s
92	389.272815s
102	425.438678s
112	431.564346s
122	495.392632s
132	571.83204s
142	650.932616s
152	691.042516s
162	762.804097s
172	715.613086s
182	762.58607s

Continued on next page

PCA dim	Fitting time
192	919.34667s

Table 5.3: Fitting times for the best model for each PCA dimension.

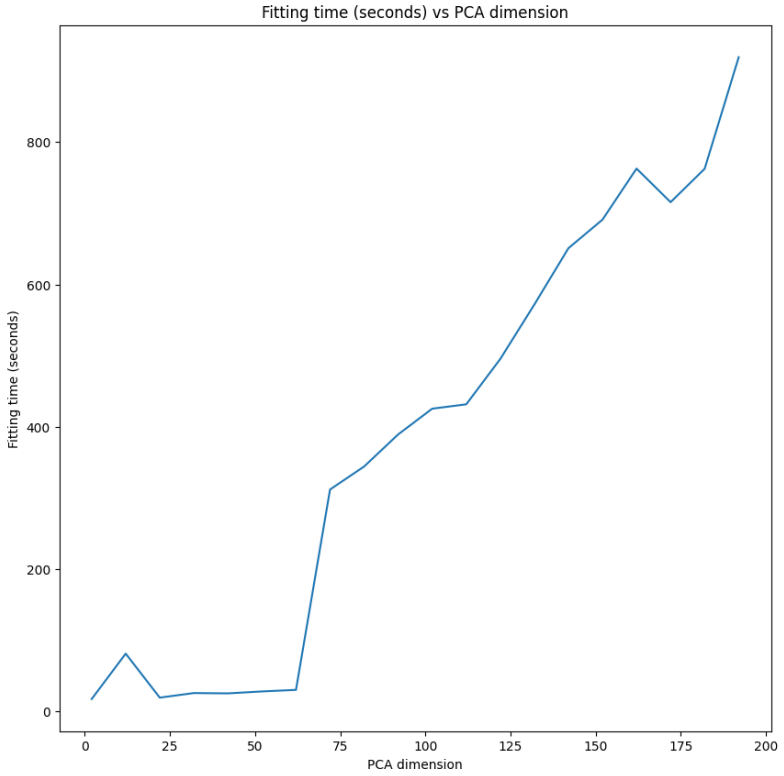


Figure 5.10: Fit time (seconds) vs PCA dimension.

As it is possible to see, the fittings times tend to increase when the PCA dimension increases: more input dimensions implies more computational time.

In Table 5.4 are reported the prediction times for the best model for each PCA dimension. In Figure 5.11 these timings are plotted vs the associated PCA dimension.

PCA dim	Prediction time
2	0.175858s
12	1.131077s
22	4.175404s
32	5.250292s

Continued on next page

PCA dim	Prediction time
42	5.720025s
52	5.911554s
62	6.041791s
72	0.390782s
82	0.500676s
92	0.616199s
102	0.745669s
112	0.864626s
122	0.984032s
132	1.108873s
142	1.21953s
152	1.303431s
162	1.405251s
172	1.485326s
182	1.57217s
192	1.643695s

Table 5.4: Prediction times for the best model for each PCA dimension.

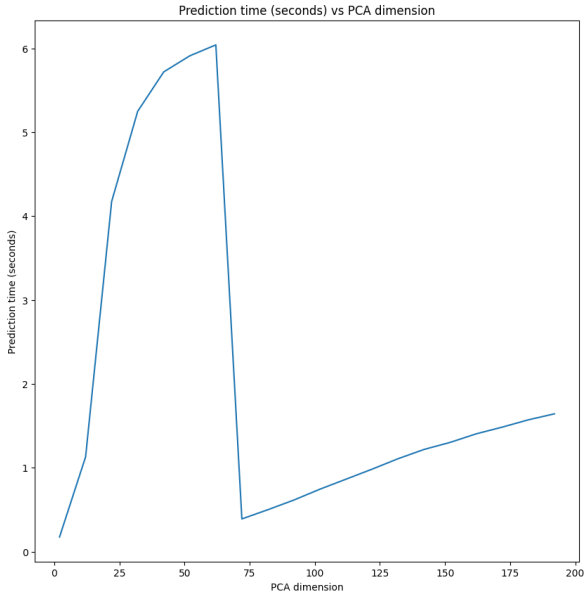


Figure 5.11: Fit time (seconds) vs PCA dimension.

As it is possible to see, the prediction times tend to increase when the PCA dimension increases: more input dimensions implies more computational time. The timings follow the

same curve trend of the 'n_clusters vs PCA dimensions' plot, implying that the prediction time is strongly correlated with the number of clusters, as expected.

Chapter 6

Normalized Cut

6.1 Theoretical Background

Let $G = (V, E, w)$ be a weighted graph, where

- V is the nodes set;
- $E \subseteq V \times V$ is the edges set;
- $w : V \times V \to R$ is a function that given two nodes returns the weight of the edges between the two nodes.

Let (A, B) be a graph "cut", where $A, B \subseteq V$ and $B = V \setminus A$. It is defined as: [16]

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

Among all possible cuts, the minimum cut is the one which minimizes $cut(A, B)$ [16]. An example is reported in Figure 6.1.

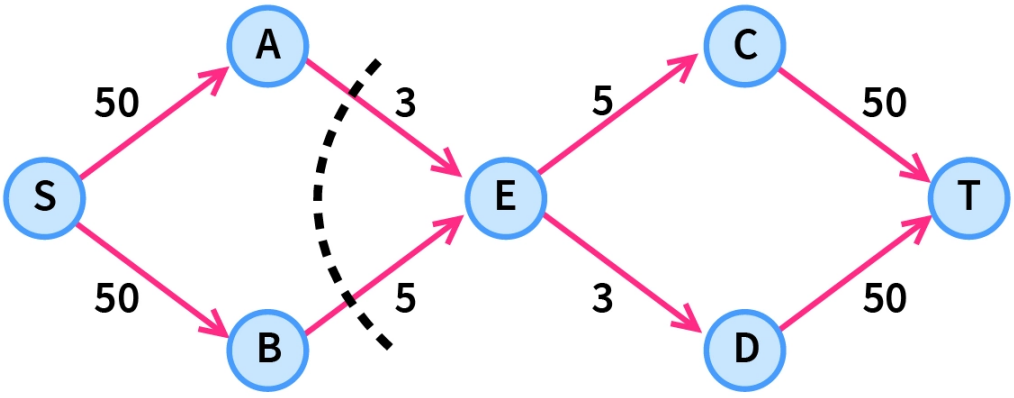
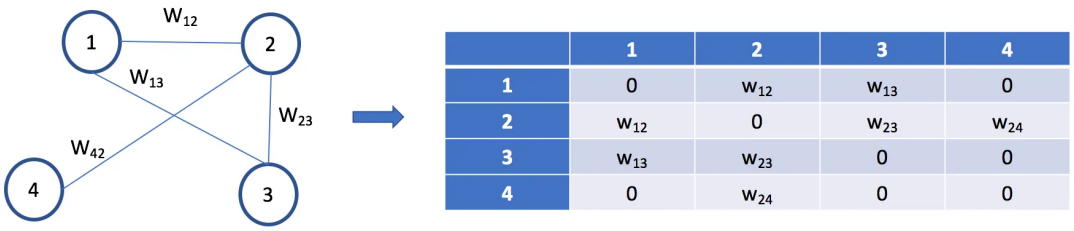


Figure 6.1: Example of minimum cut of a graph [23].

6.1.1 Graph-based clustering

Let clustering data be n vectors in $\mathbb{R}^m$ where n is the number of observations and m is the number of features. It is possible to represent these as an undirected (weighted or unweighted) graph, where each node is an observation and each edge represent the similarity between two observations [24]. This graph can be also represented with a adjacency matrix, which has the similarity between each vertex as its elements [24]. Figure 6.2



	1	2	3	4
1	0	w_{12}	w_{13}	0
2	w_{12}	0	w_{23}	w_{24}
3	w_{13}	w_{23}	0	0
4	0	w_{24}	0	0

Figure 6.2: Example of matrix representation of a weighted graph [24].

This process is done in several ways, the most common ones are:

- The ϵ-neighborhood graph: all points (vectors) whose pairwise distances are smaller than ϵ are connected, otherwise they are not connected [24]. The resulting graph is unweighted, thus every adjacency matrix's entry is set to 1 to represent a connection and to 0 otherwise [24].
- KNN Graph: use the K-Nearest Neighbors algorithm to connect node i with node j if j is among the k-nearest neighbors of i [24]. The resulting graph is unweighted, thus every adjacency matrix's entry is set to 1 to represent a connection and to 0 otherwise [24].
- Fully connected graph: connect all points (vectors) with each other, and weight all the edges by similarity s_{ij}, calculated by a Similarity Kernel Function [24]; the most common choice is Gaussian Similarity Function: [24]

$$w_{i,j} = \exp\left\{\frac{||x_i - x_j||^2}{2\sigma^2}\right\}$$

  where σ is called the scale parameter [24]. It controls the width of the neighborhoods, similar to the parameter ϵ in case of the ϵ-neighborhood graph: [24]

  – small σ: group only nearby points;
  – large σ: group far-away points.

  The resulting graph is weighted [24].

The minimum cut of this graph identifies an optimal partitioning (clustering) of the data and it is solvable in polynomial time [25]. The problem with this approach is that it favors highly unbalanced clusters (outliers) [25] [16]. Figure 6.3.

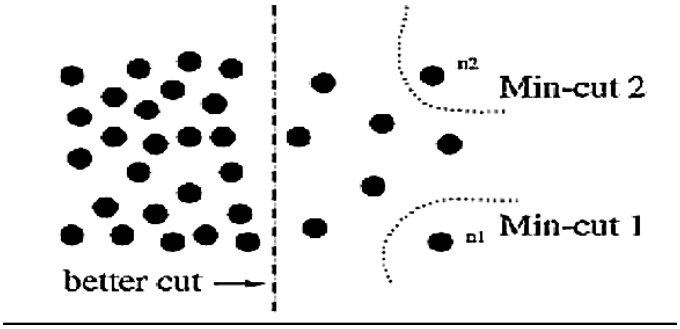


Figure 6.3: Example of unbalanced clusters [25].

6.1.2 Normalized Cut

The above problem can be avoided by using a normalized cut: [16]

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

where $Vol(A) = \sum_{i \in A} d_i$ (volume of partition A) and $d_i = \sum_j w_{i,j}$ (degree of node i) [16]. Unfortunately, identifying the minimum normalized cut is NP-hard [16].

6.1.3 Graph Laplacian

There exists an efficient approximation for the minimum normalized cut problem using linear algebra [16]. This approach is based on the Graph Laplacian (Laplacian Matrix): [16]

$$L = D - W$$

where:

- D: diagonal matrix where $D_{i,i} = d_i$ [16];
- W: adjacency matrix constructed from the graph [16].

L is symmetric (by assumption) and positive semi-defined: [16]

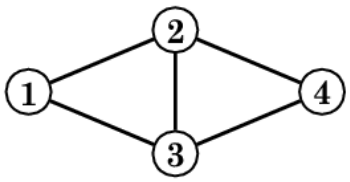
$$f^T L f = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2 \geq 0, \quad \forall f \in R^n$$

thus:

- the smallest eigenvalue of L is 0. ($\mathbf{1}$ is the corresponding eigenvector) [16];
- eigenvalues are: $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ [16].

In Figure 6.4 is reported an example of a Laplacian Matrix computation; the graph's edges are all weighted 1 and A is the similarity matrix.

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



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

Figure 6.4: Example of a Laplacian Matrix Computation [26].

Ncut as an eigensystem

Any cut (A, B) can be represented by a binary indicator vector x: [25]

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

Using the Rayleigh quotient it can be shown that: [25] [16]

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

$$\text{subject to } y^T D 1 = 0$$

Thus the problem becomes an eigenpairs problem: the Rayleigh quotient reaches its minimum value λ_{min} (the smallest eigenvalue of L) when y is v_{min} (the corresponding eigenvector) [25] [16]. Unfortunately this problem is also NP-hard, but by simply relaxing the constraint that y be a discrete-value vector and allow it to take on real values, the above problem becomes: [25] [16]

$$\min_y y^T(D-W)y \quad \text{subject to } y^T D y = 1$$

This amount to solving a generalized eigenvector problem: [25]

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

Eigenvalues of the Laplacian are approximate solutions to minimum normalized cut problem [25] [16].

6.1.4 Spectral clustering

The spectral clustering techniques implements the above idea to cluster data [27]. It requires the choice of the number of clusters k and a way to construct the adjacency matrix form vector data [27].

Spectral Clustering Algorithm:

1. Construct a adjacency matrix from vector data [27].
2. Compute the normalized Laplacian $L_{sym} = D^{-1/2}LD^{-1/2}$. This ensures that the resulting eigenvectors are invariant to rescaling and shifting of the data points [27].
3. Compute the k smallest eigenvectors $u_1, u_2, \ldots, u_k$ of L_{sym} [27].
4. Let $U = [u_1, u_2, \ldots, u_k] \in \mathbb{R}^{n \times k}$ [27].
5. Normalizes the row of U to norm 1: [27]

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

   This normalization ensures that each eigenvector has the same scale, and that the magnitude of the eigenvectors does not affect the clustering results [27].
6. Cluster the rows of U by thinking them as points in $\mathbb{R}^k$ using the k-means algorithm [27].

Spectral clustering allows to find clusters with non-convex boundaries: no assumption is made about the shape/form of the clusters [27] [16]. Figure 6.5.

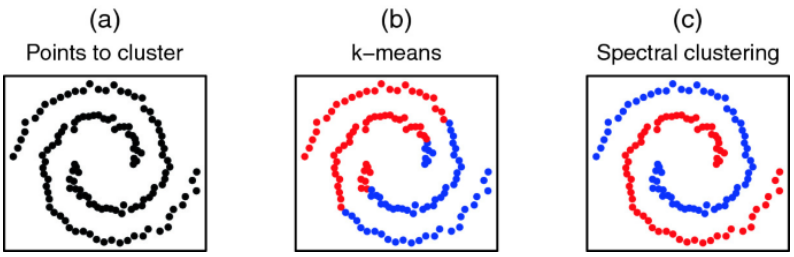


Figure 6.5: Comparison between Spectral Clustering algorithm and k-means algorithm.

Spectral Clustering Algorithm can only cluster data already seen in the fit phase [19]. This is algorithm is reasonably fast for sparse data sets of several thousand elements but it is computationally expensive for large and dense datasets [19]. This is because eigenvectors need to be computed and then clustered, a quite expensive operation [19].

6.2 Using Normalized Cut with MNIST dataset

As mentioned in Chapter 3, the project was executed on "13th Gen Intel i5-13600K (20) @ 5.100GHz" CPU and "Ubuntu 22.04.2 LTS x86_64" OS.

As mentioned in Section 3.2, the tuning parameter for the Gaussian Mixture model is the number of clusters K.

6.2.1 Results

In table 6.1 the tuning results for each PCA dimension are reported; in each sub-table the best result row is highlighted:

PCA dimension 2	
n_clusters	rand index
5	0.778585
6	0.820522
7	0.832437
8	0.836153
9	0.844401
10	0.849087
11	0.856153
12	0.862142
13	0.864235
14	0.867839
15	0.872134

PCA dimension 12	
n_clusters	rand index
5	0.719255
6	0.794991
7	0.820884
8	0.880090
9	0.878200
10	0.885281
11	0.882062
12	0.880587
13	0.899409
14	0.898147
15	0.902305

PCA dimension 22	
n_clusters	rand index
5	0.732581
6	0.808239
7	0.863521
8	0.900842
9	0.898976
10	0.912829
11	0.911494
12	0.912303
13	0.931517
14	0.926956
15	0.926088

PCA dimension 32	
n_clusters	rand index
5	0.733884
6	0.809769
7	0.864501
8	0.901571
9	0.899964
10	0.920071
11	0.921598
12	0.918209
13	0.933663
14	0.929365
15	0.928473

PCA dimension 42	
n_clusters	rand index
5	0.734568
6	0.810306
7	0.864946
8	0.901027
9	0.923036
10	0.920237
11	0.919557
12	0.916191
13	0.933880
14	0.929336
15	0.928487

PCA dimension 52	
n_clusters	rand index
5	0.735273
6	0.810133
7	0.864127
8	0.899594
9	0.921736
10	0.920075
11	0.918924
12	0.915282
13	0.933310
14	0.928856
15	0.927983

Continued on next page

PCA dimension 62	
n_clusters	rand index
5	0.777632
6	0.810623
7	0.864450
8	0.898761
9	0.921851
10	0.920157
11	0.918382
12	0.914593
13	0.932701
14	0.928245
15	0.927370

PCA dimension 72	
n_clusters	rand index
5	0.784624
6	0.810529
7	0.864121
8	0.898312
9	0.921505
10	0.919295
11	0.918260
12	0.914191
13	0.932228
14	0.927792
15	0.927129

PCA dimension 82	
n_clusters	rand index
5	0.784851
6	0.810367
7	0.863668
8	0.897164
9	0.920360
10	0.918109
11	0.917539
12	0.913605
13	0.931461
14	0.927231
15	0.926373

PCA dimension 92	
n_clusters	rand index
5	0.790697
6	0.810568
7	0.863539
8	0.897083
9	0.920027
10	0.917062
11	0.917620
12	0.913624
13	0.931526
14	0.927016
15	0.926259

PCA dimension 102	
n_clusters	rand index
5	0.794184
6	0.810431
7	0.863339
8	0.896549
9	0.919011
10	0.909958
11	0.917077
12	0.912992
13	0.931126
14	0.926758
15	0.925992

PCA dimension 112	
n_clusters	rand index
5	0.795736
6	0.810546
7	0.863131
8	0.896375
9	0.918705
10	0.909899
11	0.916968
12	0.912710
13	0.930886
14	0.926477
15	0.925770

Continued on next page

PCA dimension 122	
n_clusters	rand index
5	0.802204
6	0.810431
7	0.863079
8	0.896015
9	0.918481
10	0.910195
11	0.916750
12	0.912438
13	0.930831
14	0.926400
15	0.925703

PCA dimension 132	
n_clusters	rand index
5	0.802337
6	0.810301
7	0.862990
8	0.895430
9	0.917875
10	0.909939
11	0.916690
12	0.912233
13	0.930635
14	0.926161
15	0.925452

PCA dimension 142	
n_clusters	rand index
5	0.806107
6	0.810554
7	0.862822
8	0.895105
9	0.917630
10	0.918384
11	0.916380
12	0.911809
13	0.930417
14	0.926041
15	0.925372

PCA dimension 152	
n_clusters	rand index
5	0.805127
6	0.810615
7	0.862827
8	0.894642
9	0.917079
10	0.918464
11	0.916476
12	0.911794
13	0.930176
14	0.925862
15	0.925218

PCA dimension 162	
n_clusters	rand index
5	0.805803
6	0.810407
7	0.862619
8	0.894686
9	0.917149
10	0.918429
11	0.916238
12	0.911585
13	0.930158
14	0.925808
15	0.925307

PCA dimension 172	
n_clusters	rand index
5	0.807238
6	0.810478
7	0.862655
8	0.894347
9	0.916365
10	0.918348
11	0.916216
12	0.911373
13	0.930135
14	0.925688
15	0.925145

Continued on next page

PCA dimension 182	
n_clusters	rand index
5	0.806337
6	0.810464
7	0.862579
8	0.893981
9	0.916348
10	0.918670
11	0.914185
12	0.911299
13	0.930096
14	0.925669
15	0.925232

PCA dimension 192	
n_clusters	rand index
5	0.804410
6	0.810557
7	0.862485
8	0.893926
9	0.915962
10	0.918641
11	0.914114
12	0.911158
13	0.929845
14	0.925419
15	0.925242

Table 6.1: Normalize Cut tuning results.

As can be seen from the above tables, the best number of clusters is always greater than 10 (the true number of classes). This can be caused by the fact that the higher the number of cluster is, the higher the internal coherence of each cluster is. Another observation is that in the 90% of cases, the best number of components is 13. This implies that the number of cluster is uncorrelated with PCA dimension.

In Table 6.2 is reported the best "n_clusters" value for each PCA dimension:

PCA dim	best n_clusters	rand_index
2	15.000000	0.872134
12	15.000000	0.902305
22	13.000000	0.931517
32	13.000000	0.933663
42	13.000000	0.933880
52	13.000000	0.933310
62	13.000000	0.932701
72	13.000000	0.932228
82	13.000000	0.931461
92	13.000000	0.931526
102	13.000000	0.931126
112	13.000000	0.930886
122	13.000000	0.930831
132	13.000000	0.930635
142	13.000000	0.930417
152	13.000000	0.930176
162	13.000000	0.930158
172	13.000000	0.930135
182	13.000000	0.930096
192	13.000000	0.929845

Table 6.2: Best n_clusters for each PCA dimension.

The best PCA dimension is 42 (n_clusters=13.0) with a rand score of 0.933880, while the worst PCA dimension is 2 (n_clusters=15.0) with a rand score of 0.872139.

In Figure 6.6 are reported the 'best rand index vs PCA dimension' (LHS) and 'best hyperparameter value vs PCA dimension' (RHS) plots.

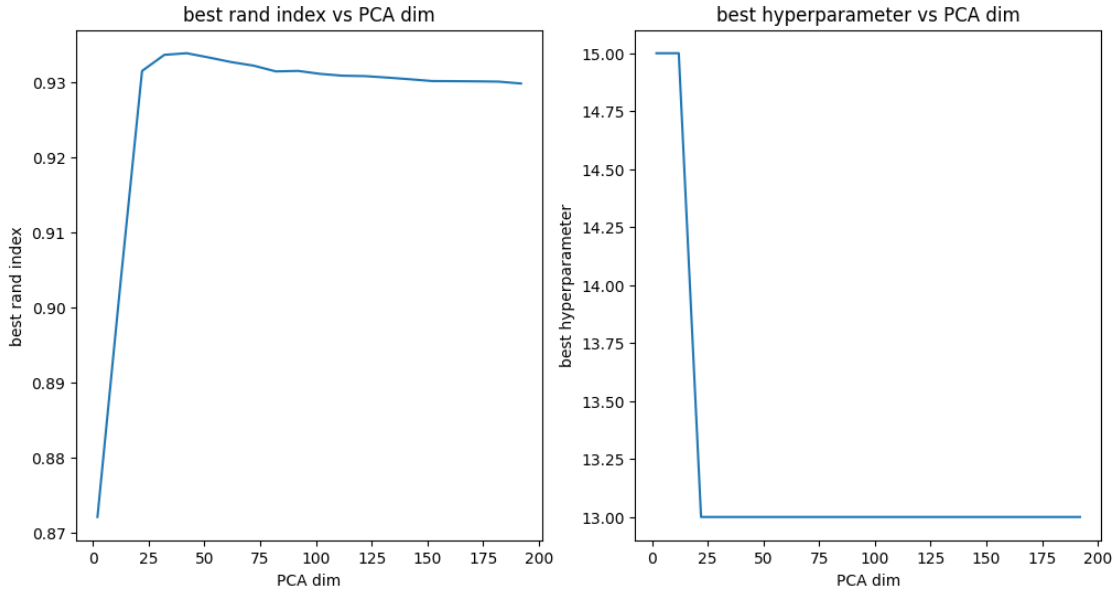


Figure 6.6: 'best rand index vs PCA dimension' (LHS) and 'best hyperparameter value vs PCA dimension' (RHS) plots for the Normalized Cut Clusterings.

As can be seen from the above table and plots:

- the rand index increases rapidly from PCA dim 2 to PCA dim 22, increases slowly until dim 42 and then it decreases slowly; This means that there is insufficient information from PCA dim 2 to PCA dim 22, while for higher dimensions the difference in information is minimal.
- the number of cluster stays 15 from PCA dim 2 to PCA dim 22 and it changes to 13 for higher dimensions; this is correlated with the rand index trend.

In Figure 6.7 is reported the plot of the best clustering for PCA dimension 2: each color and symbol correspond to a different cluster; the cluster centers are the mean of each component.

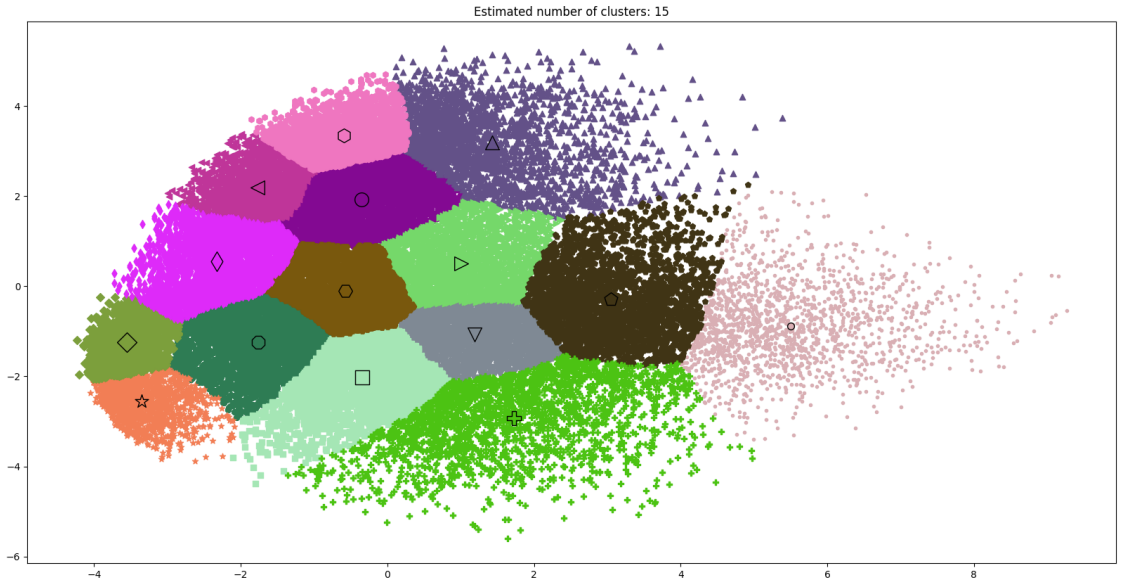


Figure 6.7: Plot of the best clustering for PCA dimension 2.

The PCA reduced data in 2 dimensions appears as an unique blob of points: there are no sharped cluster boundaries.

In Figure 6.8 is reported the cluster composition for each cluster for the best PCA dimension, while in Figure 6.9 for the worst PCA dimension: for each cluster is reported its composition in digit percentages.

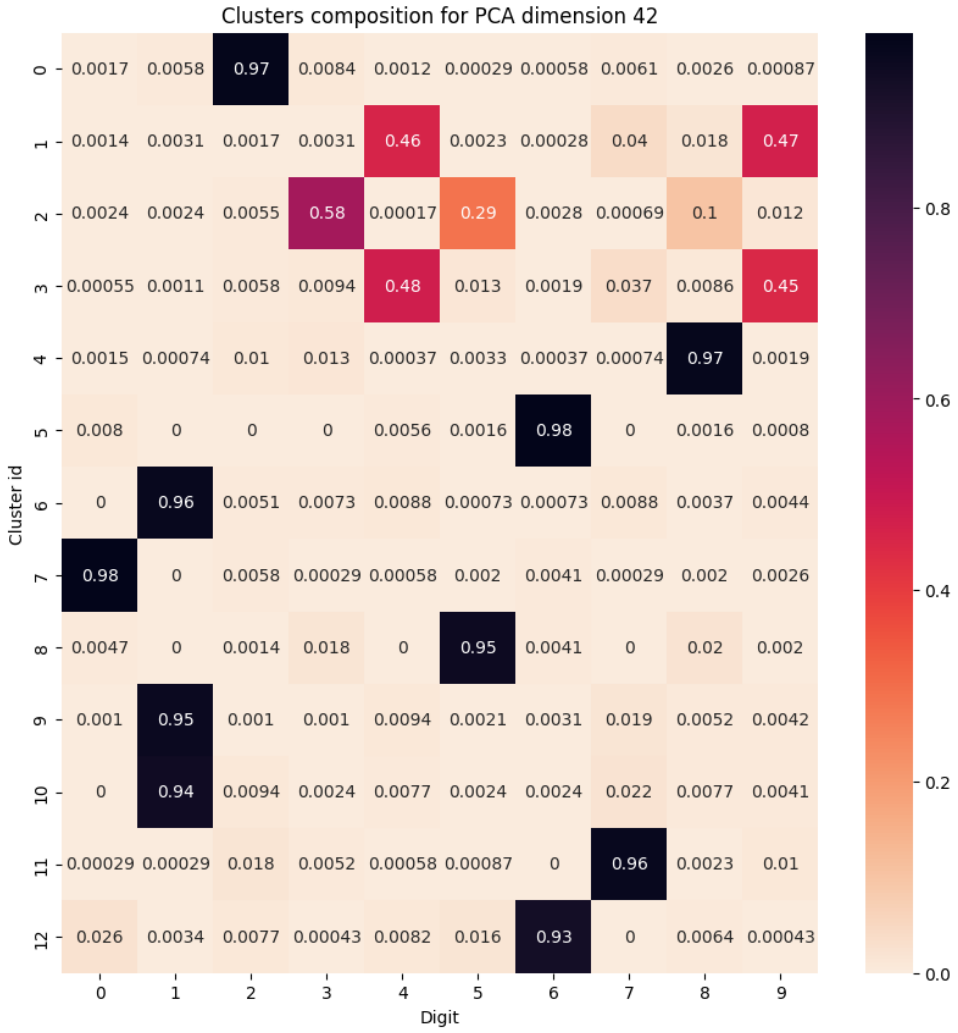


Figure 6.8: Cluster composition for each cluster for the best PCA dimension.

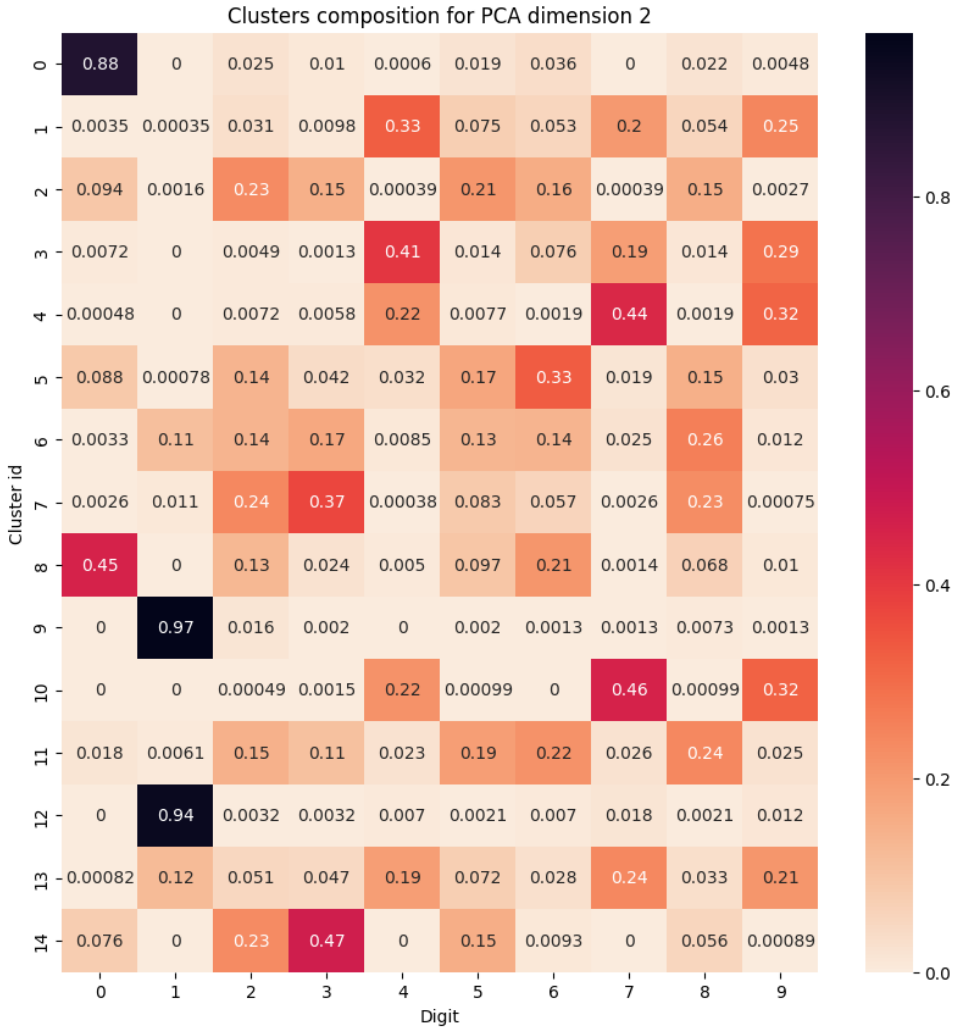


Figure 6.9: Cluster composition for each cluster for the worst PCA dimension.

As can be seen from the above figures, in the best clustering about 77% of clusters have near one digit composition, about 8% of the clusters are above half composed by one digit and about 15% of the clusters are below half or half composed by one digit; while in the worst clustering about 20% of clusters have near one digit composition, no cluster is above half composed by one digit and about 80% of the clusters are below half or half composed by one digit. The best composed clusters for the best clustering are clusters 0, 4, 5, 6, 7, 8, 9, 10, 11 and 12; while for the worst clustering are clusters 0, 9 and 12. The worst composed clusters for the best clustering are clusters 1 and 3, while for the worst clustering are clusters 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 13 and 14.

In Figure 6.10 are reported four, randomly chosen, digit images for each cluster for the best PCA dimension clustering and for the worst PCA dimension clustering; the vertical numbers are clusters ids.

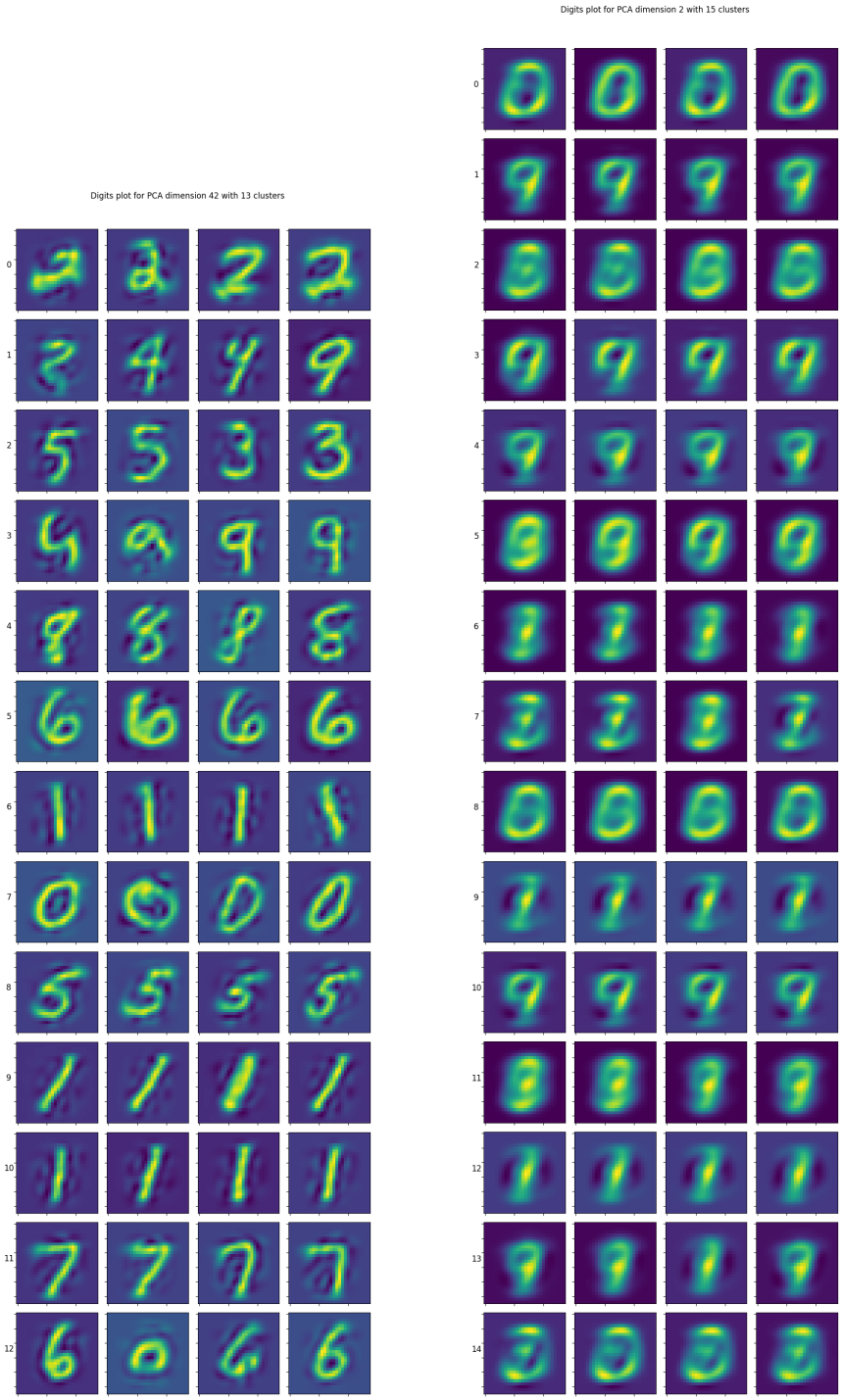


Figure 6.10: Randomly chosen digit images for each cluster for the best PCA dimension clustering (LHS) and for the worst PCA dimension clustering (RHS).

As can be seen in the above image, the best composed clusters for the best clustering contains the digits 2, 8, 6, 1, 0, 5, 7 and 6; while for the worst clustering the digits 0 and 1: this implies that the digits 0 and 1 are the simplest ones to cluster.

6.2.2 Performance

In Table 6.3 are reported the fitting times for the best model for each PCA dimension. In Figure 6.11 these timings are plotted vs the associated PCA dimension.

PCA dim	Fitting time
2	33.747574s
12	226.796395s
22	195.920918s
32	246.058957s
42	234.143907s
52	271.107374s
62	246.544036s
72	283.428245s
82	248.936403s
92	304.454295s
102	248.416887s
112	254.872454s
122	225.182179s
132	231.896396s
142	282.177064s
152	289.723123s
162	335.653323s
172	272.459099s
182	307.212695s
192	284.091749s

Table 6.3: Fitting times for the best model for each PCA dimension.

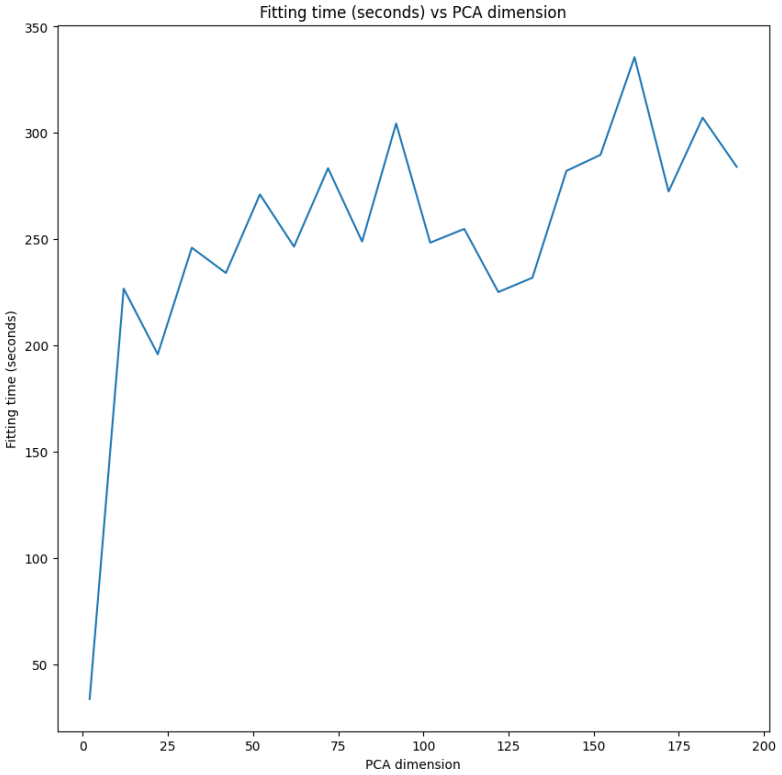


Figure 6.11: Fit time (seconds) vs PCA dimension.

As it is possible to see, the fittings times tend to increase when the PCA dimension increases: more input dimensions imply more computational time.

The prediction times for the Normalize Cut model cannot be calculated: the normalize cut algorithm can only cluster data seen during the fit phase.

Chapter 7

Conclusions

In Table 7.1 are reported the best results for each algorithm.

algorithm	best PCA dim.	best hyperpar.	number of clusters	rand index
Gaussian Mixture	12	14	14	0.8949
Mean Shift	152	5.0	5617	0.9073
Normalized Cut	42	13	13	0.9339

Table 7.1: Best results for each algorithm.

As it can be seen, the algorithm that had the best results is 'Normalized Cut', while the worst results are from the algorithm 'Gaussian Mixture'.

Both 'Normalized Cut' and 'Gaussian Mixture' require in input the number of clusters to find, while the 'Mean Shift' algorithm will estimate the number of clusters. For this specific application, when the true number of clusters is known (10 digits: from 0 to 9), the 'Mean Shift' advantage is not necessary, as the number of clusters that will give the best performance will be greater than 10: at least one cluster per digit plus some mixed clusters.

The 'Mean Shift' procedure obtained a decent score by minimizing the number of elements for each cluster; thus the number of clusters estimated is very large.

The 'Normalized Cut' and 'Mean Shift' works well with non-flat geometry [19]. For this specific application, where the clustering data is highly not flat, these two algorithms performed better compared to 'Gaussian Mixture' which works well for flat geometry.

In Table 7.2 are reported the timings to obtain the best result for each algorithm.

algorithm	fit time	mean fit time	pred. time	mean pred. time
Gaussian Mixture	0.405878s	1.736109s	0.004178s	0.014064s
Mean Shift	691.042516s	384.985351s	1.303431s	2.129767
Normalized Cut	234.143907s	251.141154	-	-

Table 7.2: Timings for each algorithm.

As it can be seen, 'Gaussian Mixture' is the fastest algorithm in both fitting and prediction, while the worst one in 'Mean Shift' in both timings. Recall that 'Normalized Cut' can only predict data already seen during the fit phase. The much higher 'Mean Shift' fitting time is partially caused by the fast that it has the higher PCA dimension of the three; furthermore the 'Mean Shift' has the higher mean fit time.

The 'Normalized Cut algorithm' gives the best result but it is almost 145 times slower than 'Gaussian Mixture' (the faster algorithm); furthermore the difference between 'Gaussian Mixture' and 'Normalized Cut' rand indexes is only 0,039. Thus, 'Gaussian Mixture' is the best compromise between results and performance.

Bibliography

[1] Yann LeCun et al. *"THE MNIST DATABASE of handwritten digits"*. URL: http://yann.lecun.com/exdb/mnist/ (visited on 12/26/2022).

[2] Patrick J. Grother. *"NIST Special Database 19 - Handprinted Forms and Characters Database"*. National Institute of Standards and Technology. URL: https://www.nist.gov/system/files/documents/srd/nistsd19.pdf (visited on 12/26/2022).

[3] Julianna Delua - IBM. *"Supervised vs. Unsupervised Learning: What's the Difference?"* URL: https://www.ibm.com/blog/supervised-vs-unsupervised-learning/ (visited on 07/29/2023).

[4] IBM. *"What is unsupervised learning?"* URL: https://www.ibm.com/topics/unsupervised-learning (visited on 07/29/2023).

[5] Wikipedia. *"Cluster analysis"*. URL: https://en.wikipedia.org/wiki/Cluster_analysis (visited on 07/29/2023).

[6] Wikipedia. *"Rand index"*. URL: https://en.wikipedia.org/wiki/Rand_index (visited on 07/29/2023).

[7] Wikipedia. *"Principal component analysis"*. URL: https://en.wikipedia.org/wiki/Principal_component_analysis (visited on 07/29/2023).

[8] Andrea Torsello - Ca' Foscari University. "Feature Selection slides".

[9] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[10] The pandas development team. *pandas-dev/pandas: Pandas*. Version 1.5.2. Nov. 2022. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[12] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[13] Casper O. da Costa-Luis. "'tqdm': A Fast, Extensible Progress Meter for Python and CLI". In: *Journal of Open Source Software* 4.37 (2019), p. 1277. DOI: 10.21105/joss.01277. URL: https://doi.org/10.21105/joss.01277.

[14] Michael L. Waskom. “seaborn: statistical data visualization”. In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[15] Oscar Contreras Carrasco. *"Gaussian Mixture Models Explained"*. URL: https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95 (visited on 07/29/2023).

[16] Andrea Torsello - Ca’ Foscari University. “Clustering slides”.

[17] Matt Bonakdarpour. *"Introduction to EM: Gaussian Mixture Models"*. URL: https://stephens999.github.io/fiveMinuteStats/intro_to_em.html (visited on 07/29/2023).

[18] -. *"Mean Shift"*. URL: https://ml-explained.com/blog/mean-shift-explained (visited on 07/29/2023).

[19] scikit-learn blog. *"Clustering"*. URL: https://scikit-learn.org/stable/modules/clustering.html#mean-shift (visited on 07/29/2023).

[20] Wikipedia. *"Kernel density estimation"*. URL: https://en.wikipedia.org/wiki/Kernel_density_estimation (visited on 07/29/2023).

[21] R.Collins. *"Mean-shift Tracking"*. URL: https://www.cse.psu.edu/~rtc12/CSE598G/introMeanShift.pdf (visited on 07/29/2023).

[22] Wikipedia. *"Multivariate kernel density estimation"*. URL: https://en.wikipedia.org/wiki/Multivariate_kernel_density_estimation (visited on 07/29/2023).

[23] Scaler Academy. *"Maximum Flow and Minimum Cut"*. URL: https://www.scaler.com/topics/data-structures/maximum-flow-and-minimum-cut/ (visited on 07/29/2023).

[24] Neerja Doshi. *"Spectral clustering"*. URL: https://towardsdatascience.com/spectral-clustering-82d3cff3d3b7 (visited on 07/29/2023).

[25] Gopalkrishna Veni. *"Normalized Graph cuts"*. URL: http://www.sci.utah.edu/~gerig/CS7960-S2010/handouts/Normalized%20Graph%20cuts.pdf (visited on 07/29/2023).

[26] Masaki Aida. *"An example of the Laplacian matrix of a simple network (n = 4)."* URL: https://www.researchgate.net/figure/An-example-of-the-Laplacian-matrix-of-a-simple-network-n-4_fig1_305653264 (visited on 07/29/2023).

[27] Wikipedia. *"Spectral Clustering"*. URL: https://en.wikipedia.org/wiki/Spectral_clustering (visited on 07/29/2023).