Università
Ca'Foscari
Venezia

[CM0623-2] FOUNDATIONS OF ARTIFICIAL INTELLIGENCE

Assignment 3:
Clustering

Victoria Grosu 881177

Contents

Chapter 1

Introduction

In this project we will use the MNIST dataset, it is a dataset of handwritten digits, composed of 70,000 images measuring 28x28 pixels. The dataset is divided into a training set of 60,000 images and a test set of 10,000 images. However for practical reasons we would only use 17000 elements.

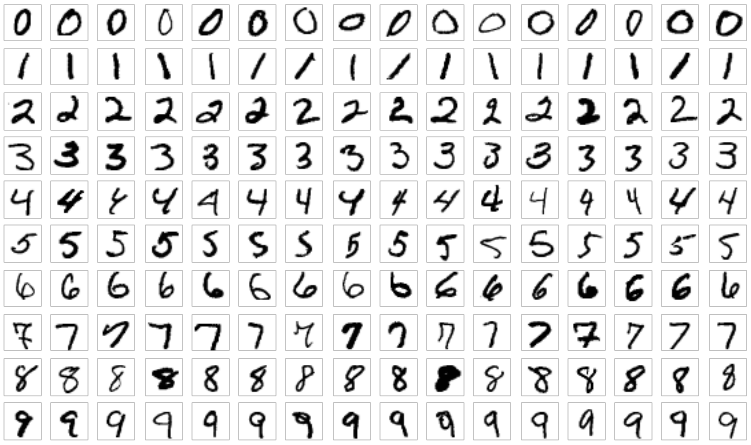


Figure 1.1: mnist_784

Unsupervised learning uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention.

In this paper we will analyze 3 types of clustering algorithms, clustering is basically a type of unsupervised learning method:

1. Mean Shift
2. Normalized Cut
3. Mixture of Gaussian with diagonal covariance

Chapter 2

Theory concepts

2.1 PCA - Principal Component Analysis

PCA is a dimensionality reduction technique that transforms a set of features in a dataset into a smaller number of features called principal components while preserving as much information as possible. *Principal Component Analysis (PCA)* assumes that the data lies on a linear subspace and helps us find the subspace.

Steps for PCA Algorithm:

1. *Standardize the data* : the first step is to standardize the data to ensure that all variables have a *mean* of 0 and a *standard deviation* of 1:

$$Z = \frac{(x - \mu)}{\sigma} \tag{2.1}$$

2. Calculate the *covariance matrix*: This matrix shows how each variable is related to every other variable in the dataset.

3. Calculate the *eigenvectors and eigenvalues*: The eigenvectors and eigenvalues of the covariance matrix are then calculated. The eigenvectors represent the directions in which the data varies the most, while the eigenvalues represent the amount of variation along each eigenvector.

4. Choose the *principal components*: The principal components are the eigenvectors with the highest eigenvalues. These components represent the directions in which the data varies the most and are used to transform the original data into a lower-dimensional space.

5. *Transform the data*: The final step is to transform the original data into the lower-dimensional space defined by the principal components.

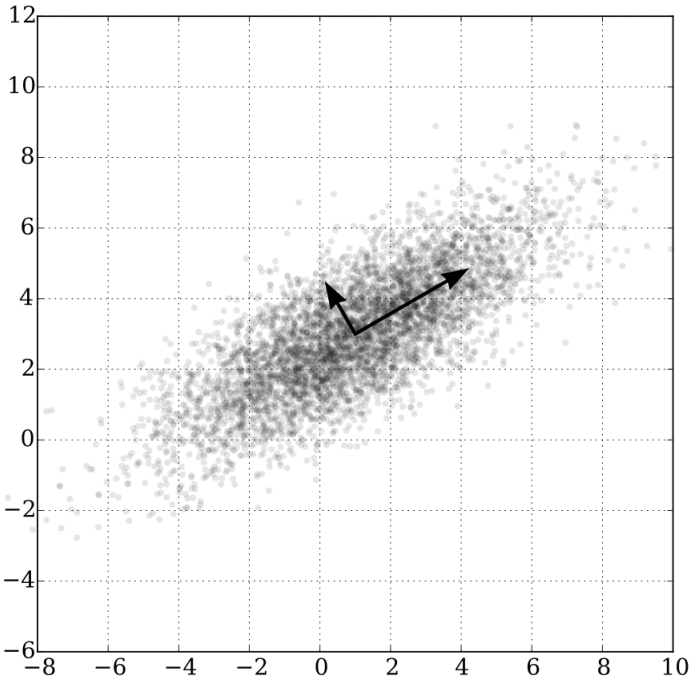


Figure 2.1: The vectors shown are the eigenvectors of the covariance matrix scaled by the square root of the corresponding eigenvalue, and shifted so their tails are at the mean.

2.2 Clustering

Clustering is basically a type of *unsupervised learning method*. An *unsupervised learning method* is a method in which we draw references from datasets consisting of input data without labeled responses.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is

basically a collection of objects on the basis of similarity and dissimilarity between them.

2.3 Mean shift

The mean shift algorithm is a *non parametric clustering* technique which does not require prior knowledge of the number of clusters, and does not constrain the shape of the clusters. The mean-shift algorithm is a hill-climbing algorithm that seeks modes of a density without explicitly computing that density.

The main idea behind mean shift is to treat the points in the d-dimensional feature space as an *empirical probability density function* where dense regions in the feature space correspond to the local maxima or modes of the underlying distribution.

For each data point in the feature space, one performs a *gradient ascent procedure* on the local estimated density until convergence. The stationary points of this procedure represent the modes of the distribution. Furthermore, the data points associated (at least approximately) with the same stationary point are considered members of the same cluster.

The cluster is created by all data points in the *attraction basin* of one mode. An *attraction basin* is the region for all the trajectories lead to the same mode.

The *Parzen windows* (or Kernels) approximate the probability density by estimating local density of points (same idea as a histogram). The kernel to be a "good kernel" has the following properties:

- Bounded
- Compact support
- Normalized
- Symmetric
- Exponential decay
- Uncorrelated

Mean Shift Clustering Algorithm:

1. Create a sliding window/cluster for each data-point.

2. Each of the sliding windows is shifted towards higher density regions by shifting their centroid (center of the sliding window) to the data-points' mean within the sliding window. This step will be repeated until no shift yields a higher density (number of points in the sliding window).
3. Selection of sliding windows by deleting overlapping windows. When multiple sliding windows overlap, the window containing the most points is preserved, and the others are deleted.
4. Assigning the data points to the sliding window in which they reside.

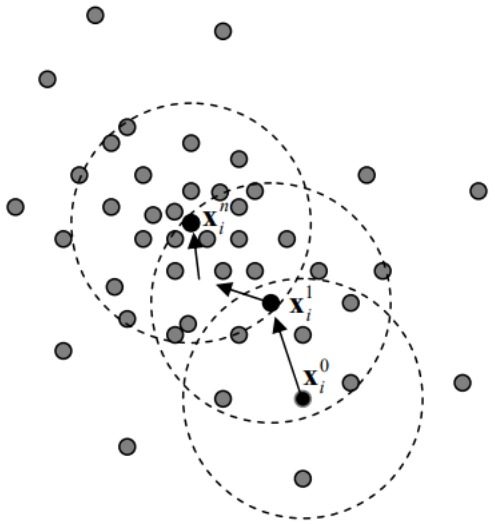


Figure 2.2: Mean shift procedure.

Properties of the Mean Shift algorithm:

- *Guaranteed convergence*: The normalization of the mean shift vector ensures that it converges.
- *Mode Detection* : Mean Shift aims to identify the modes or high-density regions in the data. Higher density regions correspond to regions with more data-points, and lower density regions correspond to regions with fewer points. It works by iteratively shifting data points towards the local mode of their sur-

rounding data points. The algorithm seeks the regions where the data density is highest.

- *Smooth Trajectory*: The angle between two consecutive mean shift vectors computed using the normal kernel is always less than 90°.

2.4 Normalized Cut

2.4.1 Clustering as graph partitioning

Let $G = (V, E)$ be an undirected graph with vertex set $V = v_1, ..., v_n$. The graph G is weighted, that is each edge between two vertices v_i and v_j carries a non-negative weight $wij \geq 0$.

A graph $G = (V, E)$ can be partitioned into two disjoint sets:

$$A \cup B = V, A \cap B = \emptyset, \tag{2.2}$$

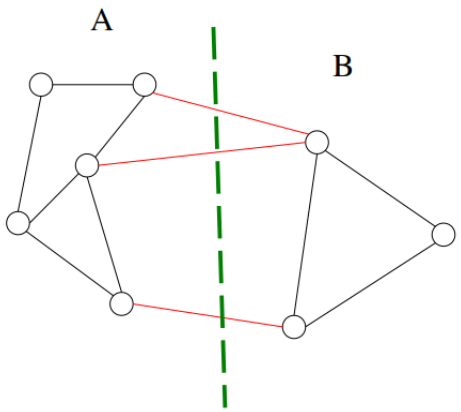


Figure 2.3: Minimum cut problem.

by simply removing edges connecting the two parts. The *degree of dissimilarity* between these two pieces can be computed as total weight of the edges that have been removed. In graph theoretic language, it is called the cut:

$$cut(A, B) = \sum_{i \in A} \sum_{j \in B} w_i j \tag{2.3}$$

The optimal bi-partitioning of a graph is the one that minimizes this cut value Fig. 2.3.

The *minimum cut* clustering is advantageous because is solvable in polynomial time but, on the other hand, it favors highly unbalanced clusters (often with isolated vertices), indeed, it only measures what happens between the clusters and not what happens within the clusters Fig 2.4.

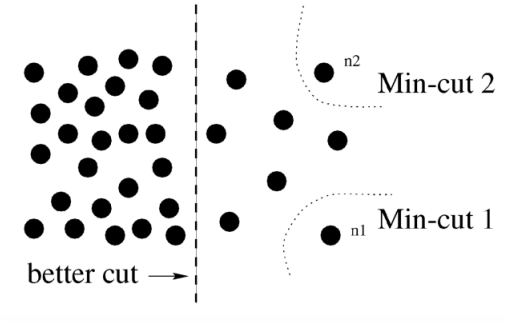


Figure 2.4: Minimum cut disadvantage.

2.4.2 Normalized Cut

In order to overcome the problem of unbalanced clusters, *Normalized Cut* is used and it is defined by:

$$Ncut(A, B) = cut(A, B) \left(\frac{1}{vol(A)} + \frac{1}{vol(B)} \right) \tag{2.4}$$

where $vol(A)$ is the volume of the set A given by $vol(A) = \sum_{i \in A} d_i$, $A \subseteq V$ and $d_i = \sum_j w_{i,j}$ is the degree of nodes (sum of weights).

Normalized cut has the advantage of taking into consideration what happens within clusters, indeed, considering $vol(A)$ and $vol(B)$ it takes into account what's going on within A and B.

The main tools to solve the *Normalized Cut* is the **Laplacian matrix**. The **unnormalized Laplacian Matrix** (or graph Laplacian) is defined as:

$$L = D - W \tag{2.5}$$

where:

1. D is a diagonal matrix where on the main diagonal there are the degrees of our nodes.
2. W is the affinity matrix, it contains 1_s or 0_s and its diagonal elements are all 0_s.

The elements of L are given by:

$$L_{i,j} = \begin{cases} d(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

where $d(v_i)$ is the degree of the vertex i.

2.4.3 Solving Normalized Cut

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

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

It can be proved that the following function can solve the *Minimum Normalized Cut* problem:

$$min_x Ncut(x) = min_y \frac{y^T(D-W)y}{y^T D y} \tag{2.6}$$

subject to: $y^T D 1 = \sum_i y_i d_i = 0$ (with $y_i \in 1, -b$).

The Normalized Cut problem is known to be *NP-hard*. This means that there is no known efficient algorithm to find the optimal solution to the Ncut problem in polynomial time for all instances.

If we relax the constraint that y be a discrete-valued vector and allow it to take on real values, the problem is equivalent to:

$$min_y \; y^T(D-W)y \quad \text{subject to } y^T D y = 1 \tag{2.7}$$

This amount to solve a *generalized eigenvalue problem*:

$$(D - W)y = \lambda Dy \tag{2.8}$$

The different steps of the Normalized Cut algorithm:

1. Represent the image as a weighted graph $G = (V, E)$, compute weight of each edge and summarize in D and W.
2. Solve $(D-W)y = \lambda Dy$ for the eigenvector with the second smallest eigenvalue.
3. Use the entries of the eigenvector to bi-partition the graph.

Why the second smallest eigenvalue? The smallest eigenvalue of Laplacians is always 0 (corresponds to the trivial partition $A = V$, $B = \{\}$)

We start from an *NP-Hard* problem and through relaxation we reach a feasible solution. However,we are not guaranteed that the relaxed solution is in one to one correspondence.

Through the relaxation we loose some precision in the final solution.

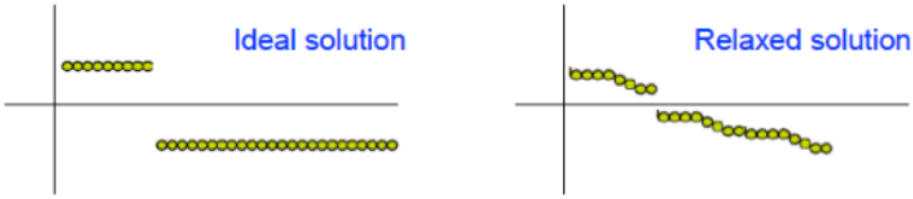


Figure 2.5: The effect of relaxation.

Note that the original problem returns binary values $(-1, 1)$, indicating the clustering membership. The relaxed version, on the right, returns continuous values of it can be the case that some points are not so clear to assign (close to the margin between the two).

2.5 Mixture of Gaussians with diagonal covariance

A Gaussian mixture model 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.

The k-means clustering relatively easy to implement, and can be applied in quite a number of use cases. But there are certain drawbacks and limitations that we need to be aware of: At each iteration, every data point is assigned uniquely to one, and only one, of the clusters. For data points that lie roughly midway between cluster centres, it is not clear that the hard assignment to the nearest cluster is the most appropriate. Adopting a probabilistic approach, we obtain 'soft' assignments of data points to clusters in a way that reflects the level of uncertainty over the most appropriate assignment. This probabilistic formulation brings with it numerous benefits.

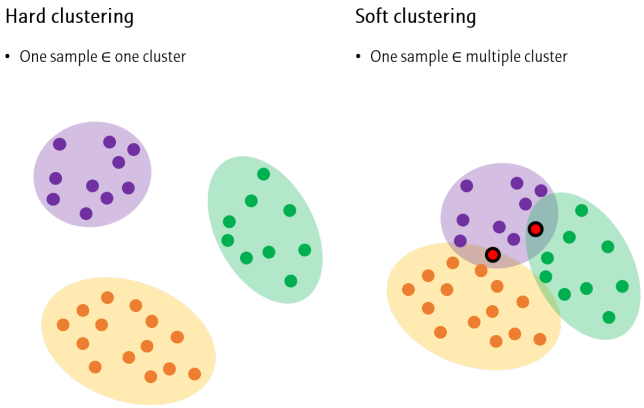


Figure 2.6: Hard Clustering and Soft Clustering. In hard clustering, one data point can belong to one cluster only. But in soft clustering, the output provided is a probability likelihood of a data point belonging to each of the pre-defined numbers of clusters.

2.5.1 Gaussian Mixture Model (GMM)

A mixture of n random variables X_i according to the *multinomial mixture* π is a random variable Y that samples data-points according to the following rule: Sample index k from π and then sample the point from X_k. In our algorithm, we are interested in the mixture of K (number of clusters) Gaussians:

$$p(x) = \sum_{k=1}^{K} \pi_k N(x|\mu_k, \Sigma_k) \tag{2.9}$$

where$\pi_k = P[\pi = k]$.

Let z be a *binary indicator variable* (cluster assignment). We can specify it in terms of the mixing coefficient $\pi_k : p(z_k = 1) = \pi_k$. Now we can express our distribution as:

$$p(x) = \sum_{x} p(z)p(x|z) \tag{2.10}$$

where

$$p(x|z) = \prod_{k=1}^{K} N(x|\mu_k, \Sigma_k)^{z_k} \tag{2.11}$$

With these last passages we reach a situation in which our clustering problem is formalized as an *optimization problem* where we want to maximize our probability to be in a cluster. In order to resolve the optimization problem, we need to introduce a very well-known algorithm: the Expectation Maximization (EM) Algorithm

2.5.2 EM algorithm

The *Expectation Maximization (EM) algorithm* is an iterative optimization algorithm that is widely used for finding maximum likelihood estimates in the presence of missing or hidden data. The EM algorithm consists of two main steps, the E-step and the M-step, which are iteratively performed until convergence.

1. The E-Step: The E-step is a computation of the expected value of the *latent variables* (or missing data) given the observed data and the current estimate of the model parameters. It is called the E-step because it computes the expectation of the *latent variables*. The expected value of the missing data or *latent variables* is known as the "posterior" distribution, denoted by $q(Z)$, where Z represents the set of all the *latent variables*. The formula for the *posterior*

distribution can be written as:

$$q(Z) = p(Z|X, \theta) \tag{2.12}$$

where $p(Z|X, \theta)$ is the conditional probability of Z given the observed data X and the current estimate of the model parameters θ.

2. The M-Step: The M-step is a computation of the maximum likelihood estimates of the model parameters given the observed data and the expected values of the latent variables obtained in the E-step. It is called the M-step because it maximizes the likelihood function. The maximum likelihood estimate of the model parameters is denoted by θ and is computed as follows:

$$L(q, \theta) = argmax(\sum_{z}(q(Z) \cdot ln\frac{p(X, Z|\theta)}{q(Z)})) \tag{2.13}$$

where $p(Z|X, \theta)$ is the joint probability of the observed and latent variables.

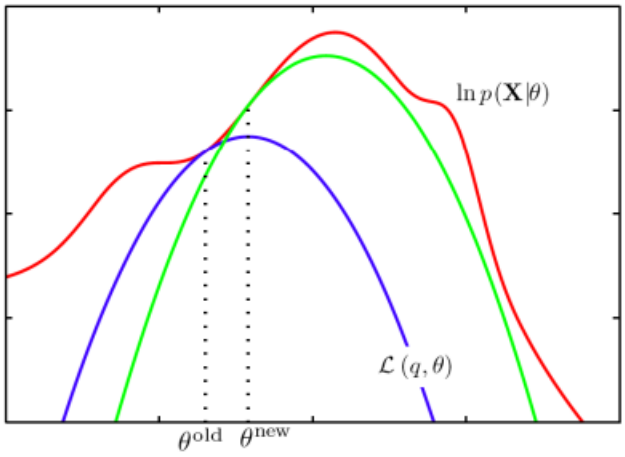


Figure 2.7: The EM algorithm involves alternately computing a lower bound on the log likelihood for the current parameter values and then maximizing this bound to obtain the new parameter values.

The EM algorithm alternates between the E-step and the M-step until convergence is achieved, i.e., until the likelihood function no longer improves or a specified number of iterations have been performed (when *Kullback-Leibler divergence* between q and p is 0).

2.5.3 EM Algorithm for GMM

1. Initialize the means μ_k , covariances Σ_k and mixing coefficients π_k
2. Iterate until convergence:
   (a) E-step: Evaluate the responsibilities given current parameters

$$\gamma(z_{nk}) = \frac{\pi_k N(x_n|\mu_k\Sigma_k)}{\sum_{j=1}^{K} \pi_j N(x_n|\mu_j, \Sigma_J)} \tag{2.14}$$

   (b) M-step: Re-estimate the parameters given current responsibilities

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk} x_n) \tag{2.15}$$

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk} x_n)(x_n - \mu_k)(x_n - \mu_k)^T \tag{2.16}$$

$$\pi_k = \frac{N}{N_k} \tag{2.17}$$

   with $N_k = \sum_{n=1}^{N} \gamma(z_n k)$
3. Evaluate log likelihood and check for convergence

$$lnp(X|\pi, \mu, \Sigma) = \sum_{n=1}^{N} ln \left(\sum_{k=1}^{K} \mu_k N(x^{(n)}|\mu_k, \Sigma_k) \right) \tag{2.18}$$

2.5.4 Things to Note

:

One drawback of GMM is that there are lots of parameters to learn, therefore may require lots of data and iterations to get good results. An unconstrained model with K -mixtures (or simply K clusters) and Ddimensional data involves fitting D×D×K+D×K+K parameters (K covariance matrices each of size D×D , plus K mean vectors of length D , plus a weight vector of length K). That could be a problem for datasets with large number of dimensions (e.g. text data), because with the number of parameters growing roughly as the square of the dimension, it

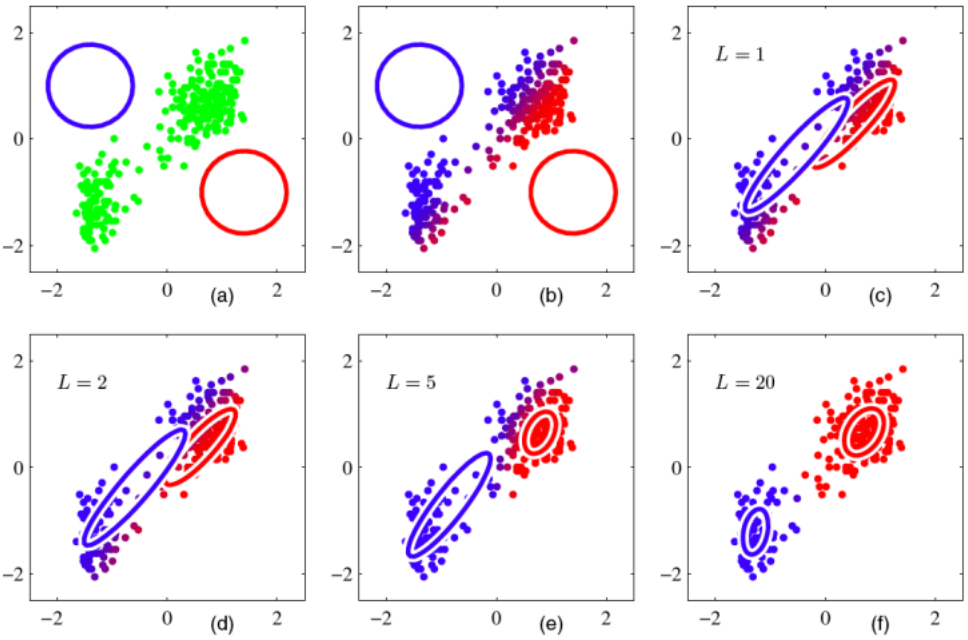


Figure 2.8: **Mixture of Gaussians vs. K-means** EM for mixtures of Gaussians is just like a soft version of K-means, with *fixed priors and covariance*. Instead of hard assignments in the E-step, we do *soft assignments* based on the softmax of the squared Mahalanobis distance from each point to each cluster. Each center moved by weighted means of the data, with weights given by soft assignments. Note that in K-means, weights are 0 or 1.

may quickly become impossible to find a sufficient amount of data to make good inferences. So it is common to impose restrictions and assumption to simplify the problem (think of it as introducing regularization to avoid overfitting problems).

One common way to avoid this problem is to fix the covariance matrix of each component to be diagonal (offdiagonal value will be 0 and will not be updated). To achieve this, we can change the covariance_type parameter in scikitlearn's GMM to diag.

Chapter 3

Evaluation

We analyze two main methods to measure the performance of the cluster algorithms:

- *Learning Time*: time needed for the algorithm to complete the clustering.
- *Rand Score*: measure of the similarity between two clusterings. It compares the clustering assignments of pairs of samples between the predicted clustering and the true labels (in our case, fortunately, available). The Rand score ranges from 0 to 1, with 1 indicating identical clusterings, and 0 indicating that the two clusterings are completely different.

$$R = \frac{2(a+b)}{n(n-1)} \tag{3.1}$$

3.1 Principal component analysis (PCA) results

PCA, or Principal component analysis, is the main linear algorithm for dimension reduction often used in unsupervised learning. This algorithm identifies and discards features that are less useful to make a valid approximation on a dataset.

Advantages of Principal Component Analysis:

1. *Removes Correlated Features*: Multicollinearity, or high correlation between independent variables, can make it difficult to determine the effect of individual variables on the predicted outcome. PCA can simplify the interpretation of the model by reducing the multicollinearity in the dataset.

2. *Improves Algorithm Performance*: When we use the principal components of the data set instead of all the variables and want to implement machine learning algorithms, this will help them to converge faster. With fewer features, the training time of the algorithms will decrease.

3. *Reduces Overfitting*: Overfitting mainly occurs when there are too many variables in the dataset. So, PCA helps in overcoming the overfitting issue by reducing the number of features.

4. *Improves Visualization*: It is very hard to visualize and understand the data in high dimensions. The mnist784 dataset is 784 dimensional which is hard to visualize, in the [Fig. 3.1] we reduced the dataset to a 2D dimension.

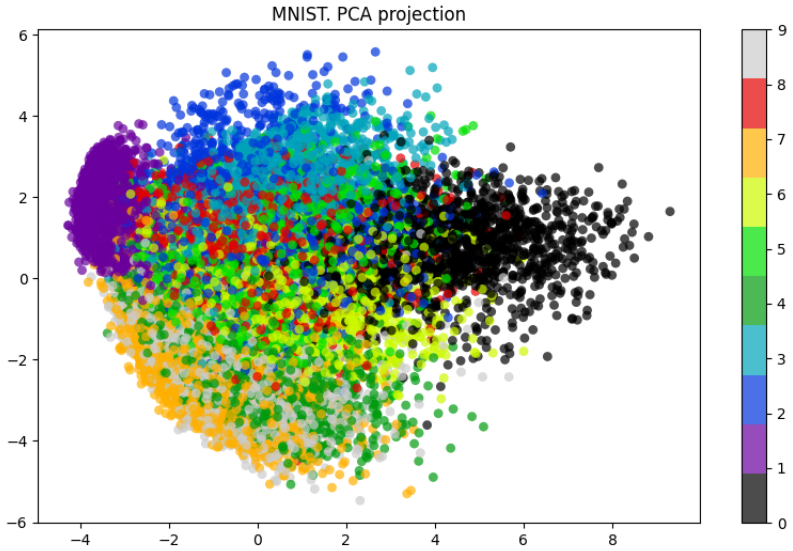


Figure 3.1: 2D PCA

Disadvantages of Principal Component Analysi:

1. *Information Loss*: AIf we don't select the number of Principal Components with care, it may miss some information as compared to the original list of features. This can lead to oversimplification or distortion of the data, and make it harder to identify outliers or anomalies. As can be seen from the images printed from the mnist784 dataset, the [Fig. 3.2] ,[Fig. 3.3] and [Fig. 3.4] , the $PCA = 2$and $PCA = 3$ show that their image reductions are similar to each other. This is because information was lost with the size reduction.

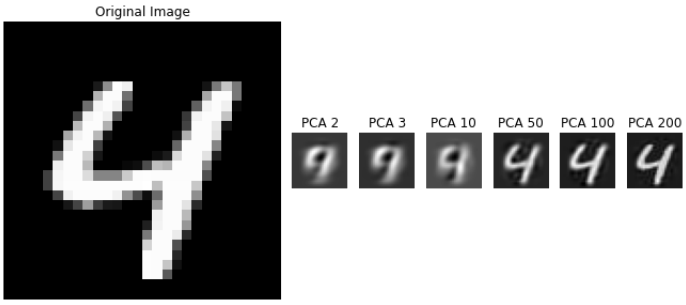


Figure 3.2: 2D PCA of number 4

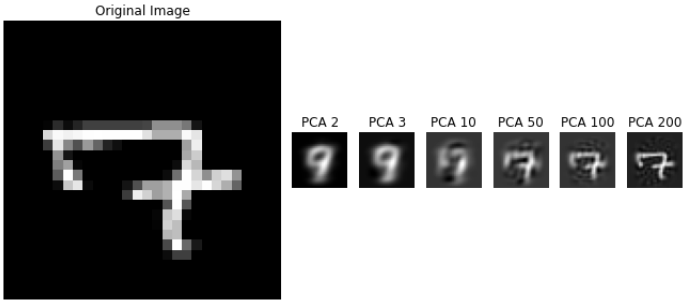


Figure 3.3: 2D PCA of number 7

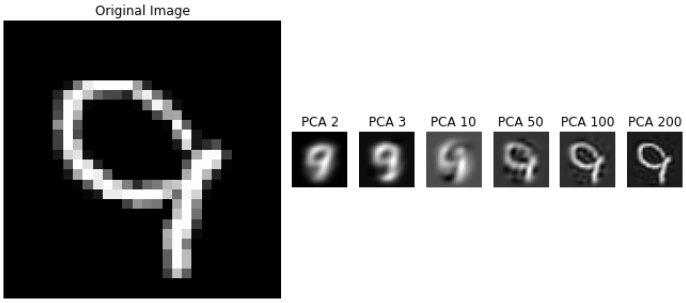


Figure 3.4: 2D PCA of number 9

2. *Independent variables become less interpretable*: After implementing PCA on the dataset, your original features will turn into Principal Components. Principal Components are the linear combination of your original features. Principal Components are not as readable and interpretable as original features.

In this paper for each clustering problem the same number of PCA number of com-

ponents will be used, so as to be able to better compare the results:

n_components	2	5	15	50	100	200

Code that will be used to compute PCA:

```
# PCA dimensional reduction
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X)
```

3.2 Mean Shift results

Mean shift clustering aims to discover "blobs" in a smooth density of samples. It is a centroid-based algorithm, which works by updating candidates for centroids to be the mean of the points within a given region. These candidates are then filtered in a post-processing stage to eliminate near-duplicates to form the final set of centroids. In our program Mean Shift clustering use a **flat kernel**.

As we have said in the section [2.3], one of the advantages of *Mean Shift Clustering* is that we don't have to choose the number of clusters. However we have to tune the 'bandwidth' parameter, so we perform a brute-force approach where you try Mean Shift clustering with different bandwidths and select the one that results in the most meaningful and interpretable clusters for each PCA number of components.

PCA	1	2	3	4	5
2	0.6	0.7	0.8	0.9	1.0
5	2.1	2.3	2.5	2.7	2.9
15	3.5	3.7	3.9	4.1	4.2
50	4.0	4.3	4.5	4.7	4.9
100	4.5	4.7	4.9	5.1	5.3
200	4.7	5.0	5.3	5.5	5.7

The model is defined by using *MeanShift* class then we fit and predict it for each PCA dimension, using the different bandwidths.

```
ms = MeanShift(bandwidth=bandwidth, bin_seeding=True)
y_pred = ms.fit_predict(X_pca)
```

When the **"bin_seeding"** parameter is set to True, initial kernel locations are not locations of all points, but rather the location of the discretized version of points, where points are binned onto a grid whose coarseness corresponds to the bandwidth.

Setting this option to True will speed up the algorithm because fewer seeds will be initialized.

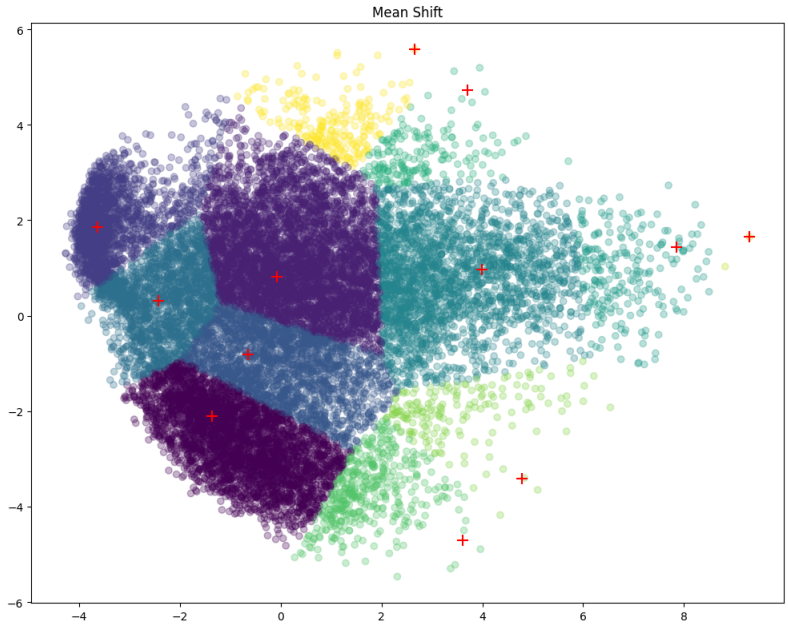
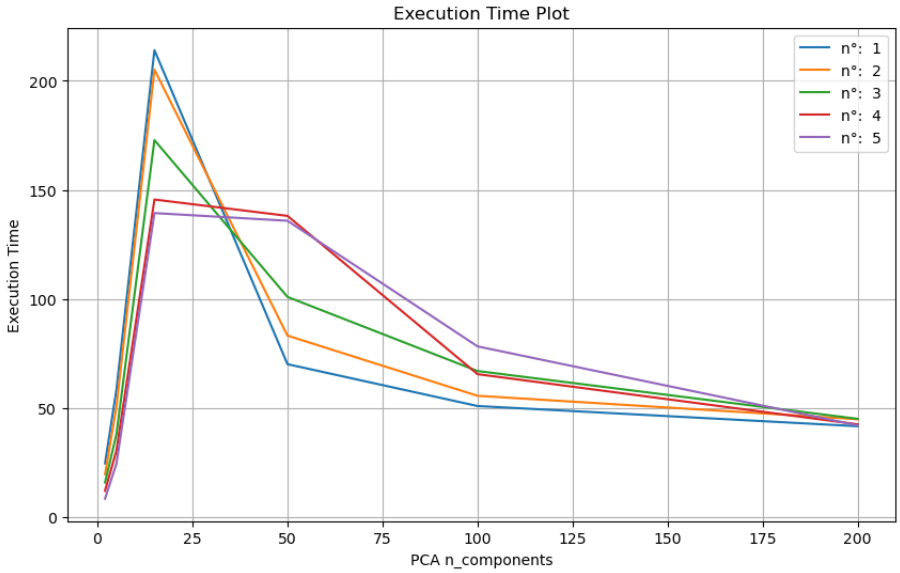


Figure 3.5: Visual representation of the Mean Shift clustering applied other the data set with bandwidth = 0.7 and PCA number of components = 2. The Mean Shift algorithm found 12 cluster centers. The round score = 0.82. When the data is projected onto a lower-dimensional space, it can be easier to visualize and understand the clustering results, especially in 2D space.

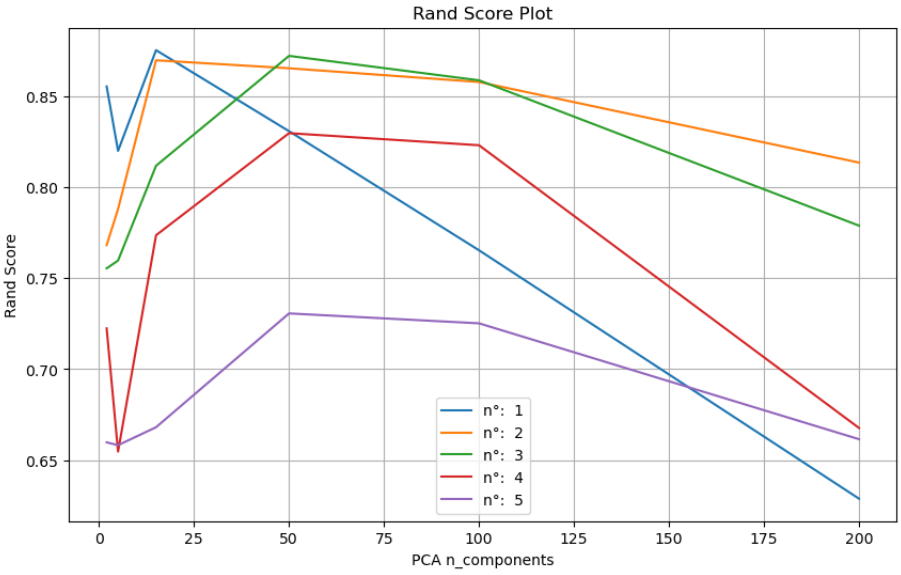
3.2.1 Learning Time

Reducing dimensionality can lead to faster convergence of the Mean Shift algorithm and potentially more interpretable clusters. However we see an anomaly in the graph where $PCA = 15$, the execution time is high compared to all other PCA dimensions. The learning time in Mean Shift clustering should not inherently be high due to a low number of PCA components. The choice of a low number of PCA components should not directly lead to increased learning time in Mean Shift clustering. The choice of initial kernel locations plays a crucial role in the performance and efficiency of clustering algorithms like Mean Shift. Mean Shift involves estimating the data density around each point, and this process can be computationally expensive for



large datasets with complex density structures. The choice of kernel and kernel bandwidth can affect the computational complexity.

3.2.2 Rand Score



In some cases, the Mean Shift algorithm may benefit from reduced dimensionality.

The reduced data can help mitigate the curse of dimensionality and make the Mean Shift algorithm more effective at identifying the modes of the data. PCA can help remove noise and irrelevant features from the data. This can lead to improved clustering results by focusing on the most informative dimensions.

We can see how the bandwidth must be set for each number of PCAs, because it depends a lot on the size of the dataset.

3.2.3 Considerations

The final result:

Best PCA number of components	15
Bandwidth	3.5
Number of Clusters	44
Best Rand Score	0.875
Execution Time	214.04 s

The advantages of Mean-shift :

1. It does not need to make any model assumption as like in K-means or Gaussian mixture.
2. It can also model the complex clusters which have non convex shape.
3. It only needs one parameter named bandwidth which automatically determines the number of clusters.
4. There is no issue of local minima as like in K-means.
5. No problem generated from outliers (thanks to the concept of moving points towards higher density regions).

The disadvantages of Mean-shift :

1. in high-dimensional spaces, the "curse of dimensionality" can affect the performance of Mean Shift. Clusters might become sparse and harder to detect due to the increased distance between points.
2. we do not have any direct control on the number of clusters.
3. it cannot differentiate between meaningful and meaningless modes.
4. choosing an appropriate bandwidth (kernel width) is critical. A bandwidth that is too small might result in too many small clusters, while a large bandwidth

might lead to over-smoothing and merging of clusters.

5. it's convergence can be sensitive to the initial positions of points, which can sometimes lead to convergence to suboptimal local modes.

3.3 Normalized Cut

Normalized Cut is a graph-based clustering technique that leverages the eigenvalues and eigenvectors of a similarity matrix to group data points together. The basic idea behind Normalized Cut is to transform the data points into a lower-dimensional space using the eigenvectors of the graph Laplacian matrix. Then, the k-means algorithm is applied to the transformed data points to obtain the final clustering.

As we know *Normalized Cut* requires knowing the number of clusters in advance, the number of clusters varies from 2 to 15.

PCA is used to reduce the dimensionality of the data before constructing the similarity graph. High-dimensional data can lead to dense and computationally expensive similarity graphs . We use **rbf_kernel** to compute an *affinity matrix*.

```
affinity_matrix = rbf_kernel(X_pca, gamma=0.1)
```

While Scikit-Learn's *SpectralClustering* class does not implement the exact Normalized Cut (Ncut) algorithm, it performs a similar task using spectral clustering techniques. We can use the SpectralClustering class to perform Normalized Cut. The model is defined by using *SpectralClustering* class then we fit and predict it on the similarity matrix calculated for each PCA dimension. Since the affinity matrix is the adjacency matrix of a graph, this method can be used to find normalized graph cuts.

```
nc = SpectralClustering(affinity = 'precomputed', n_clusters=k)
y_pred = nc.fit_predict(affinity_matrix)
```

3.3.1 Learning Time

Spectral clustering can be computationally expensive for large datasets, and the runtime may increase as the number of clusters increases because it affects the complexity of the eigenvector decomposition and the clustering step. PCA reduces the dimensionality of the data by transforming it into a lower-dimensional space while preserving as much variance as possible. Since spectral clustering involves computing similarity matrices, eigenvalues, and eigenvectors, these computations become faster in the reduced-dimensional space, leading to reduced learning times.

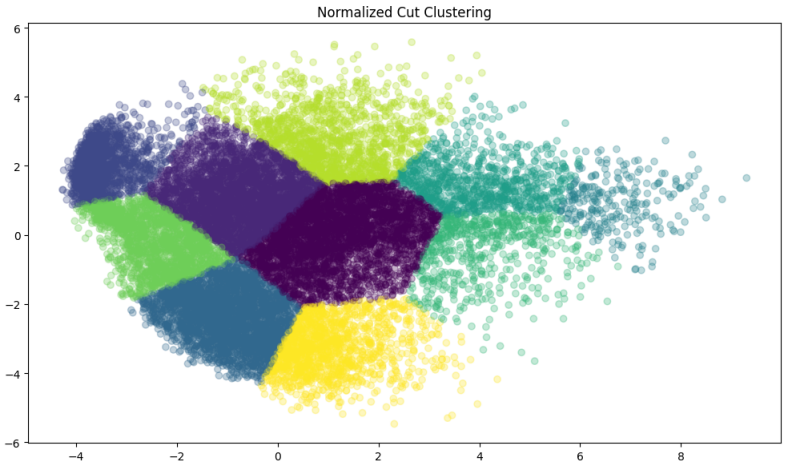


Figure 3.6: Spectral Clustering: PCA = 2, k = 10

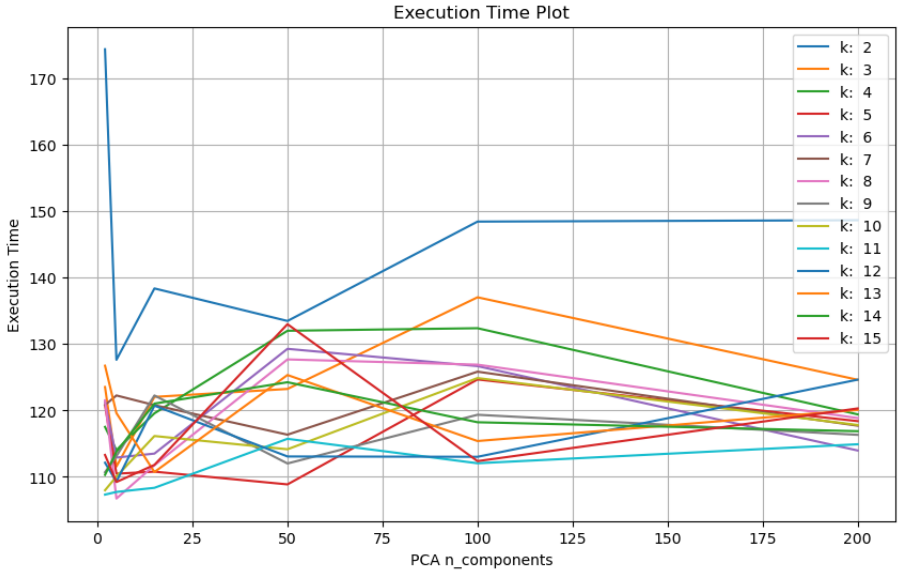
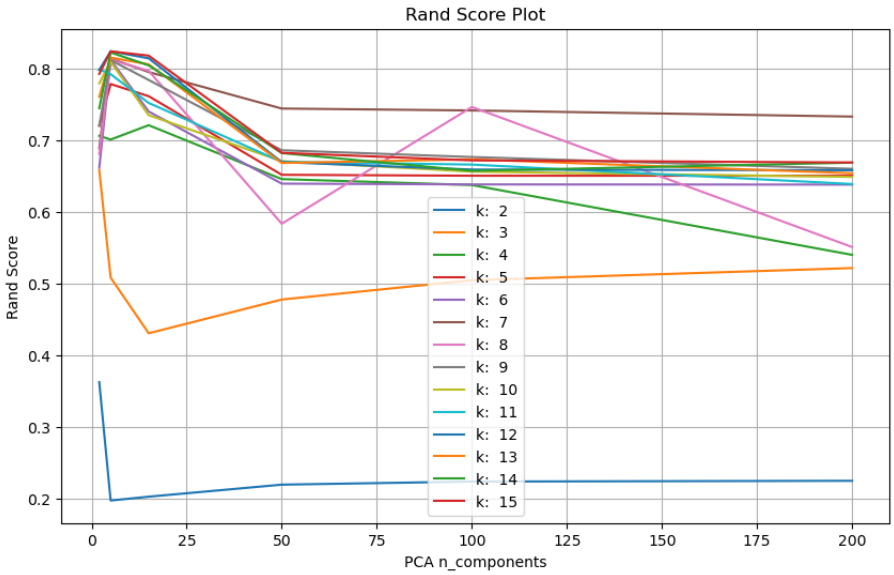


Figure 3.7: Note: The computation time of the affinity matrix is not present in the learning time. As the size of the dataset increases, the computation time of the affinity matrix increases.

As we can see from the graph, the learning rate of k=2 is the largest compared to the other number of clusters and gradually decreases as the number of clusters increases, this means that the learning rate also depends on k.

3.3.2 Rand Score



The Rand score obviously depends on the number of clusters. The better the number of clusters to represent our data, the more the Rand score increases. For low PCA the Rand score is superior to high PCA because it helps filter out noise and irrelevant features in the data. This can lead to better separation of clusters in the lower-dimensional space, enhancing the performance of spectral clustering.

For k=2 we can see from the graph that it has the worst rand score. As the number of clusters increases, the round score increases, until it drops again, exceeding k = 13.

3.3.3 Considerations

The final result:

Best PCA number of components	5
Best k	12
Best Rand Score	0.824
Execution Time	109.28 s

The advantages of normalized cut:

1. its flexibility and adaptability to different types of data and similarity measures. This is because it only requires a graph representation of the data, with nodes

representing data points and edges representing similarities.

2. it is robust to noise and outliers since it does not rely on predefined centers or distances.
3. it can handle clusters of various shapes and sizes, as it does not assume any particular geometry or distribution of the data
4. it can discover clusters that are not well separated or nested, as it does not depend on a threshold or a hierarchy to determine the number of clusters.

The disadvantages of normalized cut:

1. It is computationally expensive and memory intensive, as it requires constructing and manipulating large matrices and solving eigenvalue problems which can be difficult with high-dimensional or large-scale data.
2. it is sensitive to the choice of the similarity measure and scaling parameter as they affect the weights of the edges and shape of the matrix which then influences the eigenvectors and clustering results.
3. it is prone to over-clustering or under-clustering without a clear criterion to determine the optimal number of clusters. The eigengap heuristic is used to look for the largest gap between consecutive eigenvalues, but this may not always reflect the true structure of the data.
4. it may produce unbalanced or degenerate clusters as it does not enforce any constraint on the size or quality of clusters, potentially reducing interpretability and usefulness.

3.4 Gaussian Mixture results

Like many clustering methods, *GMM clustering* requires you to specify the number of clusters before fitting the model. The number of clusters are ranging from 2 to 15. To test which is the best number of clusters we have to iterate over this list of k, and we need to check the best number of clusters for each PCA. PCA reduces the dimensionality of the data by projecting it onto a lower-dimensional subspace while retaining as much variance as possible.

The model is defined by using *GaussianMixture* class for each PCA dimension. The number of clusters specifies the number of components in the GMM. The parameter **covariance_type** is set to **ḋiag´**, indicating that diagonal covariance matrices

should be used. The **ḋiag´** option assumes that the covariance matrix for each cluster is a diagonal matrix, meaning that the dimensions (features) are treated independently and there are no crosscorrelations between features within the same cluster.

```
gm = GaussianMixture(n_components=k, covariance_type = 'diag',
                     random_state=0)
y_pred = gm.fit_predict(X_pca)
```

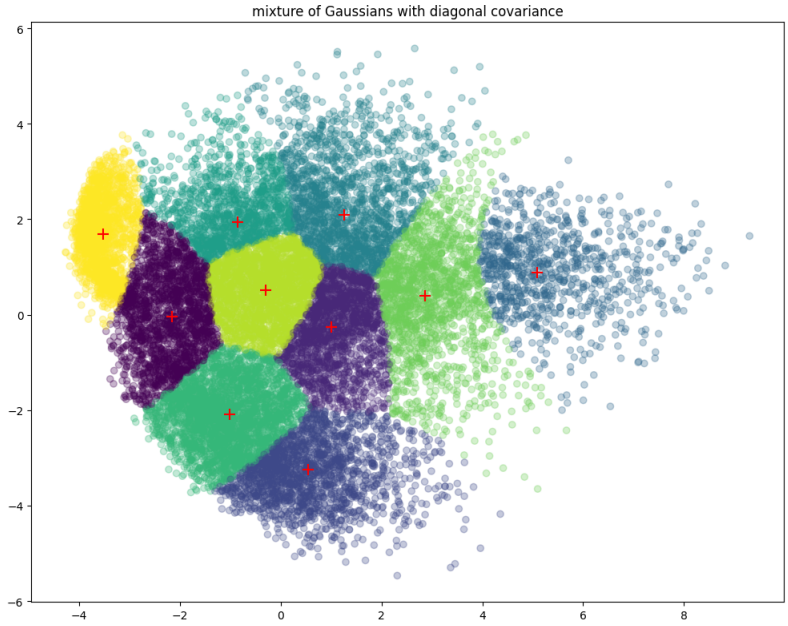
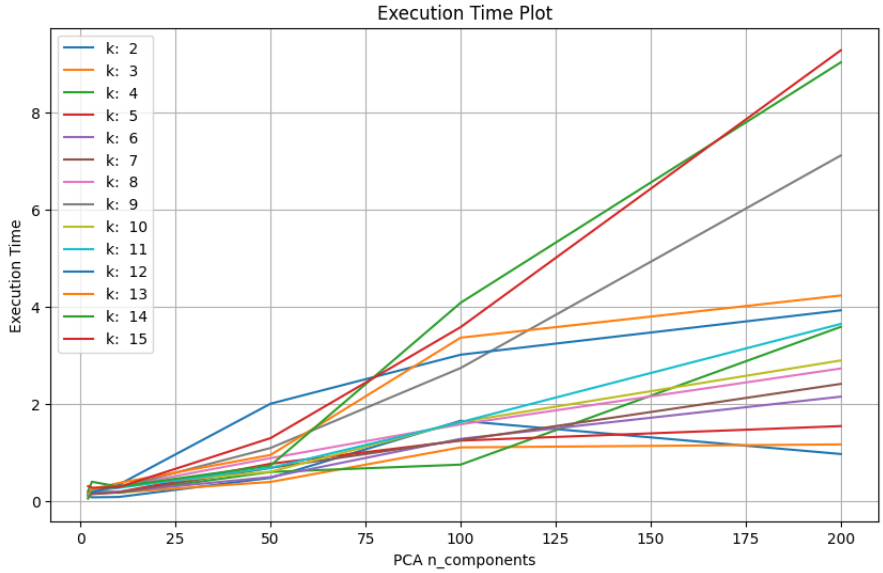


Figure 3.8: Gaussian Mixture with diagonal covariance: PCA = 2, k = 10

3.4.1 Learning Time

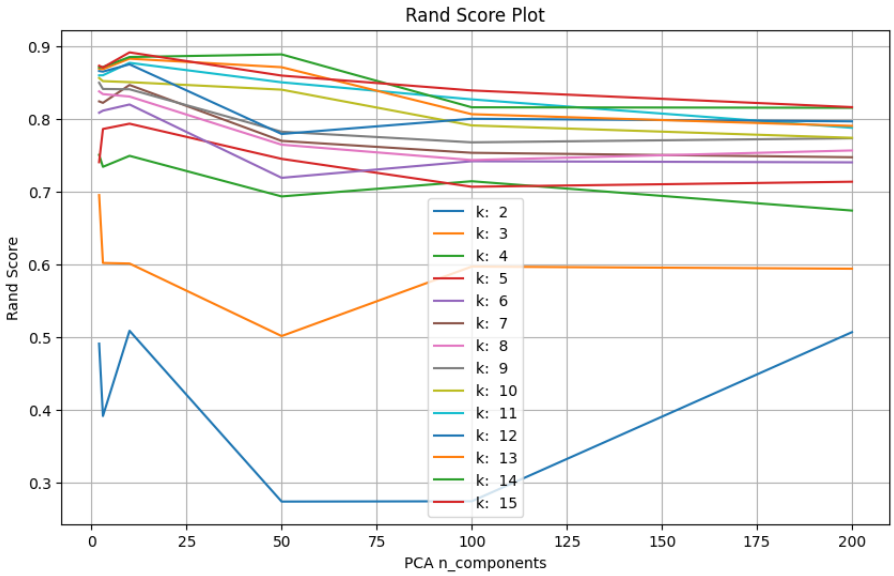
Reducing the dimensionality of the data can lead to faster convergence of the GMM algorithm, especially when dealing with a large number of dimensions. This can be particularly advantageous for large datasets. When applying a GMM with diagonal covariance matrices to PCA-transformed data, the model's complexity is further reduced. This can lead to faster convergence during training and make the model more computationally tractable.

As with SpectralClustering, the number of clusters influences the learning rate, the smaller k is the higher the execution time because the algorithm looks for Gaussian



distributions in our data.

3.4.2 Rand Score



Gaussian Mixture with diagonal covariance works very well for this dataset even without necessarily reducing its size, cause additional information does not signif-

icantly contribute to improving the clustering performance. We can use a smaller number of PCA and the round score is still good, because PCA can help remove noise and irrelevant features from the data. This can also lead to a more robust and interpretable clustering result. This suggests that a smaller number of PCA dimensions can be sufficient to capture the most important information and still achieve good results without sacrificing computational efficiency.

From the graph we see that for k=2 the round score is low and increases as the number of clusters increases. The maximum round score is reached with k = 15. So the number of clusters greatly influences the efficiency of the model.

3.4.3 Considerations

The final result:

Best PCA number of components	2
Best k	15
Best Rand Score	0.895
Execution Time	0.30 s

The advantages of Gaussian Mixture:

1. it provides a probability distribution over clusters for each point. This allows for "soft" clustering, which can be more representative of real-world data where points may belong to multiple clusters to varying degrees.
2. it provides a measure of uncertainty in the assignment of points to clusters through the probabilities. This can be particularly useful when dealing with noisy or ambiguous data.
3. it does not assume spherical clusters.
4. it handles clusters of differing sizes.
5. using **diagonal covariance matrices** assume that variables are uncorrelated. It has fewer parameters to estimate compared to full covariance matrices. This can help mitigate the "curse of dimensionality," where the number of parameters grows exponentially with the number of dimensions.
6. using **diagonal covariance matrices** can lead to faster convergence during the optimization process compared to estimating full covariance matrices.

The disadvantages of Gaussian Mixture:

1. it's performance can be sensitive to the initial placement of clusters, which can lead to convergence to local optima. Poor initialization might result in suboptimal clustering solutions.
2. it has to select the number of clusters.
3. it assumes that each cluster follows a Gaussian distribution, which might not hold true for all types of data. It might struggle with clusters of non-Gaussian shapes or data with heavy-tailed distributions.
4. the Expectation-Maximization algorithm used to estimate parameters might not always converge to the global optimum, especially if the data has overlapping clusters or the initializations are poor.
5. since we need to estimate a covariance matrix in order to use gaussian mixture models, you should make sure that you have enough data points in each cluster to adequately estimate the covariance. The amount of data required is not huge, but it is larger than simple algorithms that do not estimate a covariance matrix.

Chapter 4

Conclusion

PCA is a very common way to speed up your Machine Learning algorithm by getting rid of correlated variables which don't contribute in any decision making. The training time of the algorithms reduces significantly with less number of features. So, if the input dimensions are too high, then using PCA to speed up the algorithm is a reasonable choice.

Recap:

PCA	PCA	Best K	RandScore	ExeTime
MeanShift	15	44	0.875	214.04 s
SpectralCluster	5	12	0.824	109.28 s
GaussianMixture	2	15	0.895	0.30 s

In conclusion, the best clustering method that best analyzes our dataset is Gaussian Mixture Clustering. Gaussian Mixture has the best Rand Score, has the shortest learning time and uses only 2 PCA components and therefore consumes less memory space. Furthermore, it is close to the correct number of clusters, i.e. 10. GMMs are particularly effective when data can be accurately represented as a mixture of Gaussians, whereas Spectral Clustering and Mean Shift may struggle with certain types of data distributions for this reason they need a higher number of PCA components and more training time.

Mean Shift is good in case we don't know the correct number of clusters, however in this case we know very well that the number of clusters is equal to 10 (number of digits). Although the accuracy of the mean shift is not very far from that of the GMM, unfortunately it returns a number of clusters equal to 44. There is a risk

of overfitting the data, it cannot differentiate between meaningful and meaningless modes. In this paper we chose the bandwidths by hand, and a series of tests were done to choose the correct values. Too low bandwidths lead to a high number of clusters and vice versa. Let's say that for our problem it is not the best.

The problem with Spectral Clustering is that it is computationally expensive and memory intensive, as it requires constructing and manipulating large matrices and solving eigenvalue problems which can be difficult with high-dimensional or large-scale data. However, if you can pre-compute the affinity matrix (as was done in this paper), then the learning time can be reduced when testing. In our example, not counting the computation time of affinity matrices, only the computation of clusters with Spectral Clustering is less than the computation time of clusters with Mean Shift. Another positive aspect compared to Mean Shift is that the number of clusters i.e. 15 is close to the number of digits i.e. 10.