Università
Ca'Foscari
Venezia

Master's Degree Programme in Computer Science and Information Technology
Software Development and Engineering

3rd assignment

Artificial Intelligence: Clustering

Student
Andrea Munarin
Matriculation Number 879607

Academic Year
2022-2023

Index

Chapter 1

Introduction

The purpose of this assignment is to learn how to cluster different elements using several clustering techniques. Our main purpose is to classify and group together elements that seem similar. In order to test this, we are going to use the MNIST database[1]. This is composed of 70000 28×28 pixel gray-scale images of handwritten digits divided into 60000 training set and 10000 test set.

Figure 1.1: Example of data presents in the MNIST dataset

All the pictures can be represented as 784-dimensional (28×28) feature vectors of real values between 0 (black) and 1 (white).

[1]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. The database is also widely used for training and testing in the field of machine learning.

Once we obtain all the data, we can cluster our data with these different techniques:

- ***mixture of Gaussians*** with diagonal covariance (Gaussian Naive Bayes with latent class label);
- ***mean shift***;
- ***normalized cut***;

All the algorithms will be implemented using external library. The unsupervised classification will be performed at varying levels of dimensionality reduction through PCA[2] (going from 2 to 200) In order to asses the effect of the dimensionality in accuracy and learning time.

For the mixture of Gaussians and normalized-cut we will analyze the different results varying the number of cluster k from 5 to 15. For mean shift, instead, we are going to vary the kernel width, and for each of them we provide the value of the Rand index that we will discuss later.

[2]Principal Component Analysis describe in section 2.5

Chapter 2

Theory concepts

In this chapter, we will look at the main concepts behind "Learning" and some of the most famous technique, understanding how they work.

2.1 Clustering

We can have mainly three type of learning based on the structure of our problem:

- ***Supervised Learning***: all our data has an associate label that refers to the expected output.
- ***Unsupervised Learning***: all our data are only collection of information, and we try to group them into "natural cluster".
- ***Reinforcement Learning***: an agent interacting with the environment, makes observations, takes actions, and is rewarded or punished; The agent tries to maximize the rewards.

All the techniques we are going to look in this paper are based on "***Unsupervised Learning***".
Strong assumption that we are going to consider in some algorithms that we will see are:

- The **number of clusters is given**
- The clusters form a **partition of data** (no overlapping clusters)

2.2 Mixture of Gaussians

2.2.1 K-means

In K-means the main idea comes from the purpose to define ***cohesiveness***. In order to do so, we need to introduce some notation:

- u_k represents the center of the k-th cluster ($k = 1, ..., k$)
- $r_{nk} \in 0, 1$ is a ***binary indicator variable*** that tells us if the data point x_n is assigned to the k cluster.

Once we introduce this concepts is easy to understand that the cohesiveness of our clusters can be represented as follows:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - u_k||^2$$

K-means is based on the **optimization of** J, in particular we want to optimize J with respect to r_n and with respect to u_k. As a way to do this we proceed by minimizing J with respect to both the variable (keeping fixed the one we are not considering) iterating these two main steps:

- Optimization of r_nk:

$$r_{nk} = \begin{cases} 1, & \text{if k} = \text{argmin}_\text{j} ||x_n - u_j||^2 \\ 0, & \text{otherwise} \end{cases}$$

- Optimization of u_nk (setting derivatives to zero):

$$u_k = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}}$$

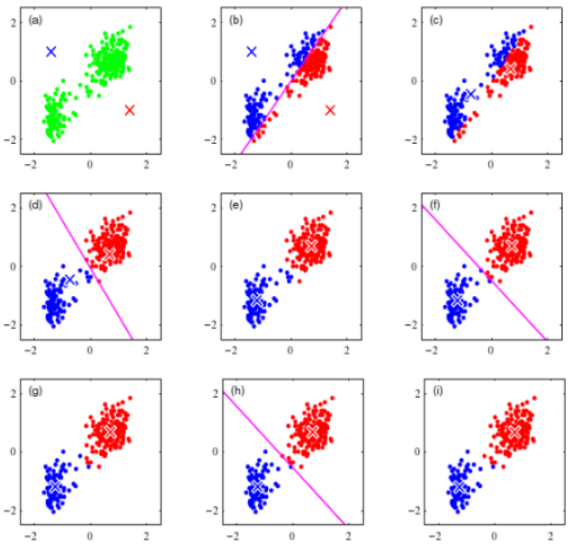
More simply the algorithm works iterating these two steps until convergence:

1. Assign the point to the **closest cluster center**
2. **Recompute cluster center** based on the current assignment for each point

K-means is base on a **descent algorithm**, and so the **convergence is assured**, but we have two main limits:

- Might converge to a local minimum
- is based on the use of squared Euclidean distance (nonrobust to outliers and find mainly spherical clusters)

Figure 2.1: K-means algorithms



2.2.2 Mixture of Gaussians

Mixture of gaussians works similar to K-means[1] but is more sophisticated and, as we will see, permits us to **achieve better solutions**.
For mixture of gaussian we adopt a probabilistic approach in which a **point has a certain probability to appertain to the k-th clusters**. As the name suggest, our model is based on the Gaussian distribution, so we need to consider both the mean of our cluster but also the variance. Now we need to define what a mixture is.

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:

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

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

[1] indeed K-means will result as a limit case of Expectation Maximization Algorithm for a Gaussian mixture model

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

$$p(x) = \sum_x p(z)p(x|z)$$

where[2]

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

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.

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**. It is particularly useful in problems where the data has a complex structure, and some variables are not observed or are **latent variables** (In our scenario the latent variable is referred to as the "***cluster assignment variable***", which indicates which cluster a particular data point belongs to). 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:

   $$q(Z) = p(Z|X, \theta)$$

   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]notice that now we have no more sums but only products

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:

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

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

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).

Summarizing, the EM algorithm is an iterative algorithm that finds the maximum likelihood estimate of the model parameters in the presence of missing or hidden data. The algorithm works by alternately computing the expected values of the missing data given the observed data and the current estimate of the model parameters in the E-step, and then updating the model parameters by maximizing the expected log-likelihood in the M-step.

Applying this algorithm to our mixture of gaussians we obtain **how to update the value**, in particular given: $\gamma(z_{nk}) = \frac{\pi_k N(x_n|\mu_k \Sigma_k)}{\sum_{j=1}^{K} \pi_j N(x_n|\mu_j, \Sigma_J)}$ and $N_k = \sum_{n=1}^{N} \gamma(z_n k)$ we have:

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

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

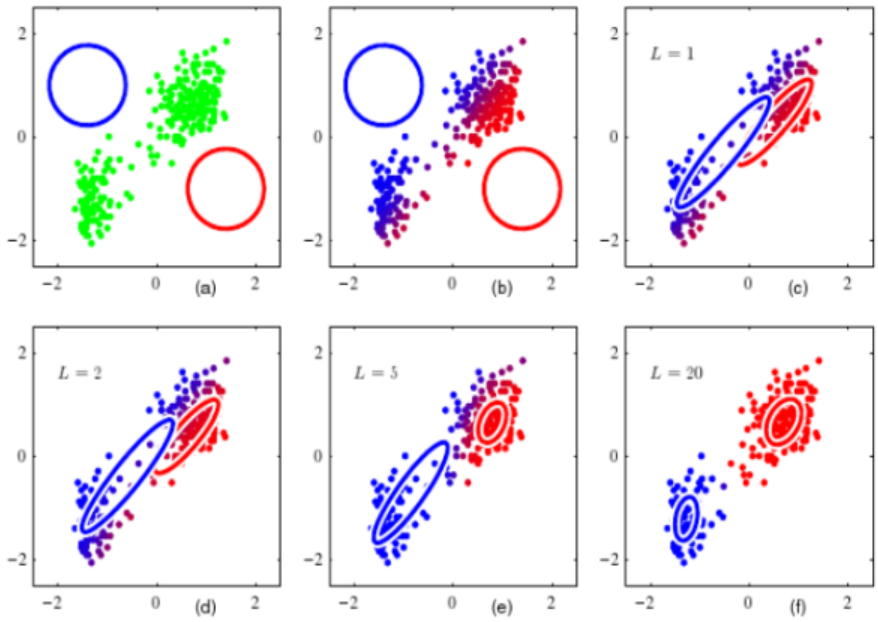
$$\pi_k = \frac{N_k}{N}$$

So the two steps can be resumed as:

1. During the E-step of the algorithm, the **posterior probabilities** of the cluster assignments are **computed for each data point**, given the current estimates of the model parameters.

2. During the M-step, the **model parameters are updated based on the posterior probabilities** and the data points that belong to each cluster.

Figure 2.2: Mixture of Gaussians algorithm



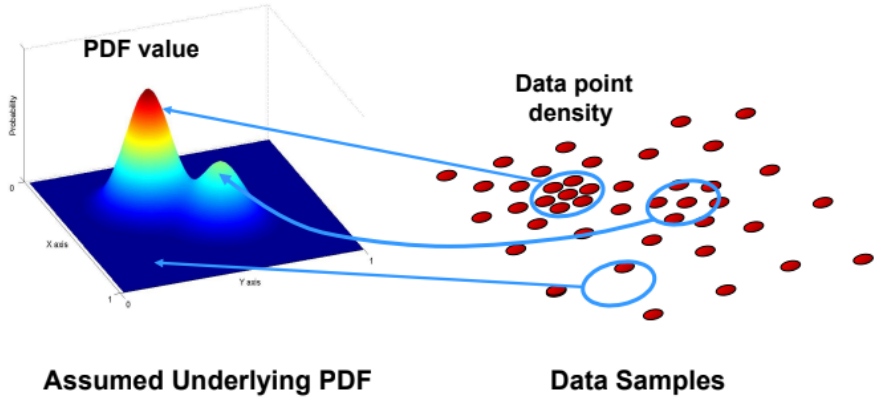
2.3 Mean shift

Mean shift is a very powerful clustering algorithms. Mean shift is 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. The main idea of the algorithm can be described in 5 points:

1. Choose a **search window size**.

2. Choose the **initial location** of the search window.
3. **Compute the mean location** (***centroid*** of the data) in the search window.
4. **Center the search window** at the mean location computed in Step 3.
5. Repeat Steps 3 and 4 until convergence.

Figure 2.3: Non-parametric Density Estimation



As you can imagine the search window (or ***kernel***) is very important for finding the optimal solution. In particular a **good kernel** should satisfy the following properties:

- ***Bounded***: not infinite size
- ***Compact support***
- ***Normalized***: $\int_{R^d} K(x)\,dx \;= 1$
- ***Symmetric***
- ***Exponential decay***: $\lim_{||x||\to\infty} ||x||^d K(x)\,dx \;= 0$
- ***Uncorrelated***

We can choose different kernel with this properties, the most famous are listed here:

- ***Flat kernel***

- Epanechnikov
- Normal
- Uniform

We will use the flat kernel in our implementation, but we will discuss better how, **varying the bandwidth**, we can achieve different results.

Now let's see some very interesting properties of Mean Shift algorithm:

- ***Guaranteed convergence***: The normalization of the mean shift vector ensures that it converges.
- ***Mode Detection***: Our algorithms move the center of our window towards the nearest mode. The set of all locations that converge to the same mode define the basin of attraction of that mode.
- ***Smooth Trajectory***: The angle between two consecutive mean shift vectors computed using the normal kernel[3] is always less than 90°.

2.4 Normalized cut

In ***Normalized cut*** we can say that our cluster problem is seen under a ***different prospective***, in particular, our main purpose now is to consider our clustering problem as a ***partition of a weighted graph***. In order to model our problem using graph representation, we need to define the main ideas:

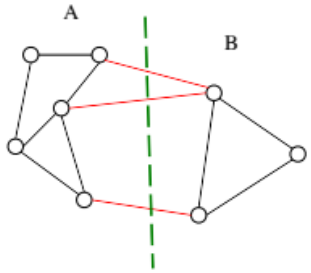
- The weighted graph represents a **similarity matrix** between the objects associated with the nodes in the graph.
- A **large positive weight** connecting any two nodes (high similarity) biases the clustering algorithm to place the nodes in the same cluster.
- The **graph representation is relational** in the sense that it only holds information about the comparison of objects associated with the nodes.

[3]In practice the convergence of mean shift using the normal kernel is very slow, and typically the uniform kernel is used.

- **Cut** divides the graph into two **subgraphs**. The cut can be calculated as follows:

$$Cut(C_1, C_2) = \sum_{i \in C_1} \sum_{j \in C_2} w_{ij}$$

Figure 2.4: Notion of Cut



A very intuitive idea can be trying to **iterativly cut** of the graphs where the value of the **cut is minimum** (different cluster must have low similarity). This approach has a well-known problem: we are **only considering relation between clusters** and not the cohesiveness of the elements inside the cluster.

A better approach would be to consider the **Normalized cut**, in which we take in account also the elements inside the subgraphs (using the notion of ***Volume***[4]) during the cut:

$$NCut(C_1, C_2) = Cut(C_1, C_2)(\frac{1}{Vol(c_1)} + \frac{1}{C_2})$$

It can be shown that the problem of minimize the Normalized Cut is **NP-Complete**, and is related to the ***Laplacian Matrix***, or graph Laplacian (defined using the diagonal matrix[5] and the affinity matrix):

$$L = D - W$$

In order to achieve the solution of this NP-hard problem we need to ***relax it***, in particular we don't consider anymore a perfect cut of the graph in which a node is in a subgraph or in the other (-1, or +1). Now, considering **real values for**

[4] $Vol(C) = \sum_{i \in C} D(x_i)$, where $D(x_i)$ represents the degree of that node

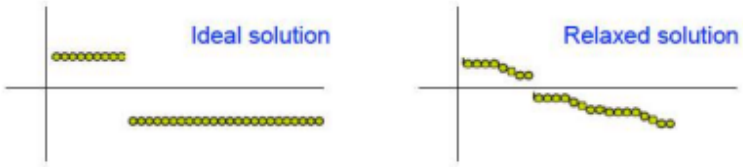
[5] Matrix in which all elements in diagonal are the degree of that node and all the other values are 0

the participation of a cluster, we can model our problem as an ***eigenvalue*** and eigenvector[6] problem:

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

So, computing the **second-lowest eigenvalue** (first is 0 because of the structure of Laplacian matrix), the corresponding eigenvector contains the cluster indicator for each data point.

Figure 2.5: Relaxed Solutions



It is also possible, using one of several existing techniques, to find k clusters using the k smallest eigenvalues (and their associated eigenvectors).

2.5 PCA and Feature Selection

PCA, or ***Principal Component Analysis***, is a widely used technique for feature selection and **dimensionality reduction** in machine learning and data analysis. The general formula for PCA can be described as follows:

1. ***Standardize the data***: the first step is to center the data around its mean and scale it by its standard deviation.

2. Compute the ***covariance matrix***: The covariance matrix is a measure of the linear relationship between pairs of variables.

3. Calculate the ***eigenvectors*** and ***eigenvalues*** of the covariance matrix: The eigenvectors are the **directions in which the data varies the most**, while the eigenvalues represent the amount of variance explained by each eigenvector.

4. ***Sort the eigenvectors by their corresponding eigenvalues***: the eigenvectors are sorted in descending order based on their eigenvalues, with the

[6] very powerful properties that are also independent of the permutation of the matrix

eigenvector with the highest eigenvalue being the first principal component, the eigenvector with the second-highest eigenvalue being the second principal component, and so on.

5. Select the ***top k principal components***: The top k eigenvectors with the highest eigenvalues are selected, where k is the number of features or dimensions to retain.

6. ***Project the data onto the new feature space***: The selected k eigenvectors are used to transform the data into a new feature space with a reduced dimensionality.

The result of PCA is a set of transformed features that capture the most important patterns in the original data while reducing the number of features to a smaller, more manageable subset. The selected principal components can be used for further analysis or as input features for machine learning algorithms.

After projecting the data onto the new feature space, it is possible to ***reconstruct*** the original data by projecting it back to the original feature space using the selected principal components. However, since the number of features is reduced, the **reconstructed data will not be identical to the original data**. The difference between the original data and the reconstructed data is called the ***reconstruction error***. Minimizing the reconstruction error is an important consideration in selecting the appropriate number of principal components to retain.

In the end, it's important to mention the two big **limits** of PCA:

- Finds **only linear subspaces**, but the data can be on a more complex manifold
- It is **insensitive to the classification** task

Chapter 3

Implementation

Before going through each implementation it's important to understand that all the following algorithms will follows the same principles with slight difference.

All the data were fetched only ones and **saved in a npz file** in order to save time during the testing of the code. Pay attention to the fact that the feature vector is normalized by diving each value by 255.0. At the end we obtain 70000 feature vector X of 784 real values between 0 and 1 and 70000 labels.

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

List of main important libraries:

- `sklearn`:
  - `sklearn.decomposition` in order to use the implementation for PCA
  - `sklearn.metrics` to obtain the `rand_score` function that helps us to compute the efficacy of each algorithm
  - all the clustering algorithm
- `NumPy` and `pandas` will be very useful to handle our data structures
- `time` and `matplot` will be used to collect evaluation data and plot them.

The flows of the programs are basically the same:

1. we first read the data from our npz archive

```
data = np.load('mnist.npz')
X = data['X'].astype(np.float32)
y = data['y'].astype(np.float32)
```

2. we initialize our `rand_scores` and our `learning_times` vectors to all zeros

```
rand_scores = np.zeros((len(n_components_range),
                        len(bandwidth_range)))
learning_times = np.zeros((len(n_components_range),
                           len(bandwidth_range)))
```

3. Each algorithm has been done using two nested fors, in which we go through various levels of reduction of dimensionality and different number of kernels (or kernel bandwidth)

   - dimensionality will vary from 2 to 200, but with different steps, based on required computing time
   - the number of kernel will be from 1 to 15 with a step = 1
   - the kernel bandwidth will vary as follows: [0.1, 0.5, 1, 2, 5, 10, 20, 50, 100]

4. we construct the model, measure the time before and after the call to `fit_predict`

```
gmm = GaussianMixture(n_components=k, covariance_type='diag')

t0 = time.time()
y_pred = gmm.fit_predict(X_pca)
t1 = time.time()
learning_time = t1 - t0
```

5. we compute the `rand_score`

```
rand_score_computed = rand_score(y, y_pred)
```

6. we insert the time and the score in the vectors

```
rand_scores[i, j] = rand_score_computed
learning_times[i, j] = learning_time
```

- Only for Gaussian Mixture we reconstruct using PCA inverse function, and we display the means for each cluster and only for dimensionality equal to 2, 100 or 200

```
fig, axs = plt.subplots(nrows=1, ncols=k, figsize=(5*k, 5))
for l in range(k):
    axs[l].imshow(pca.inverse_transform(gmm.means_[l])
                  .reshape(28, 28), cmap='gray')
```

7. At the end of all the work we save the graphs of times and scores, and we memorize all the information in an archive npz

```
fig, ax = plt.subplots(figsize=(10, 7))
for j, k in enumerate(k_range):
    ax.plot(n_components_range,
            rand_scores[:, j],
            label='k={}'.format(k))
ax.set_xlabel('xlabel')
ax.set_ylabel('ylabel')
ax.set_title('title')
ax.legend()
plt.savefig(GaussiansMixtureRandScore.png)

...

np.savez('GaussiansMixture.npz',
         rand_scores=rand_scores,
         learning_times=learning_times,
         n_components_range=n_components_range,
         k_range=k_range)
```

One last thing to say is that the **spectral clustering** implementation includes special instructions at the beginning of the algorithm and at the end of each step to **store the current progress** and information gathered, as the algorithm requires significant time to execute. This precaution is necessary to prevent loss of partial data in case of unexpected errors or system shutdowns.

Chapter 4

Evaluation

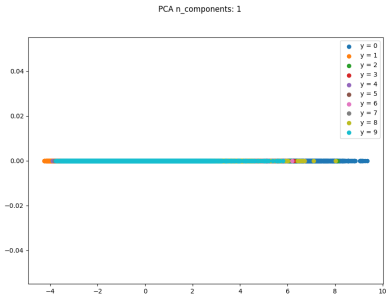
Let's start analyzing the result of each clustering algorithm singularly, and then spot the similarities and differences between each of them.

4.1 PCA results

Before going into the details of the performance of the various algorithms, we have to understand which are the result of the ***PCA*** and how we can interpret them.

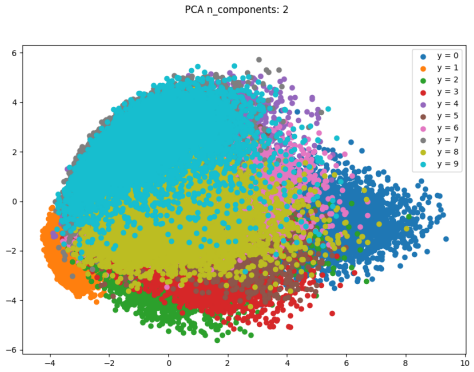
When PCA is performed on a dataset and only one dimension is retained, it means that the **data is projected onto a single line or axis**. The single dimension retained is the direction along which the data varies the most, and this direction is known as the **first principal component**. In our context of image data, this corresponds to a direction that captures the most variation in pixel intensity across the entire dataset.

Figure 4.1: PCA - 1 dimension



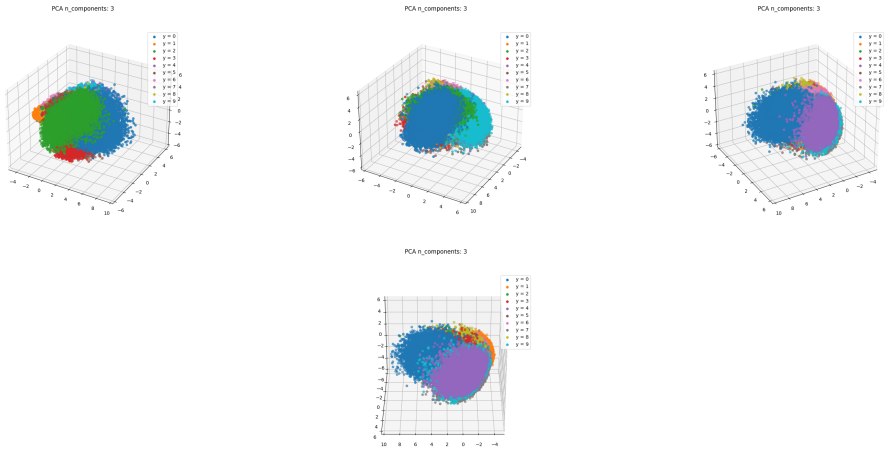
If two dimensions are retained in PCA, the data is projected onto a **two-dimensional plane**. The two dimensions retained correspond to the first and second principal components, which capture the t**wo most significant sources of variation in the dat**a. This is the easiest to understand and to visualize.

Figure 4.2: PCA - 2 dimension



When three dimensions are retained, the data is projected onto a three-dimensional space, with the **three dimensions** corresponding to the three most significant sources of variation in the data. In the figure 4.3 we can see all the vectors from several points of view.

Figure 4.3: PCA - 3 dimension



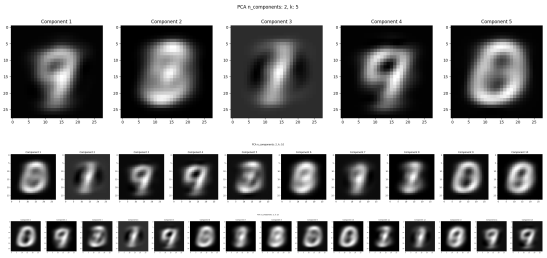
Performing PCA with 1, 2, or 3 dimensions can make it difficult to identify the ap-

propriate clusters for each item, particularly when **elements with different colors and labels overlap**. While PCA is a powerful technique for reducing the dimensionality of our feature vector space, if not executed correctly, it may yield unsatisfactory results.

4.2 Gaussian Mixture

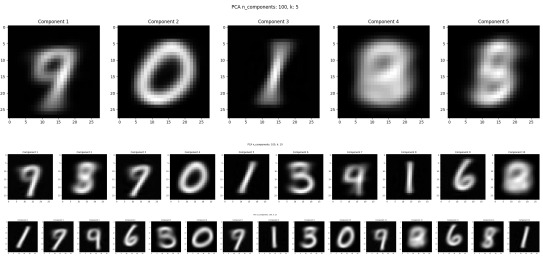
To analyze how PCA impacts the performance of algorithms, we employed the Gaussian Mixture algorithm and examined all possible values of PCA from 2 to 200, while varying the number of clusters from 5 to 15. Additionally, **we reconstructed the images for PCA dimensions 2, 100, and 200** to understand what the model had learned. Before looking at the performance, let's have a look at ***PCA reconstruction***.

Figure 4.4: PCA - 2 dimension



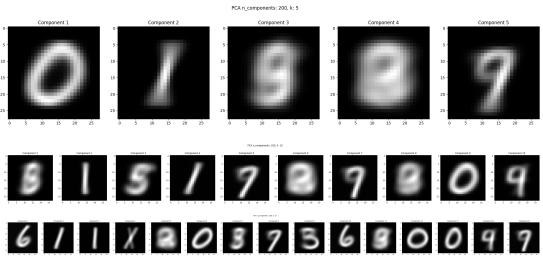
For PCA 2, as only the two most important components are considered, **some information may have been lost** in the reconstruction process. Moreover, with such little information, the clustering algorithm may not have recognized the different clusters correctly. In fact, even with **k=10 clusters, the algorithm failed to recognize some numbers** (such as 2, 4, and 5, while 7 was a mixture of 1 and 9). For k=15, we did not obtain well-defined shapes.

Figure 4.5: PCA - 100 dimension



For PCA 100, although the exact digits were not yet clustered correctly, we obtained better and more precise results (as seen in picture 4.5). Interestingly, we found that adding 100 components for **PCA 200 did not improve the quality of our clusters**, and even resulted in non-digit-shaped clusters (such as X) for k=15. This suggests that **200 may not be the optimal choice, as it does not yield significantly better results and increases computational time**[1].

Figure 4.6: PCA - 200 dimension



Now let's discuss the **performance** of the algorithm with respects to number of cluster and PCA. We analyze for each algorithm two main measures:

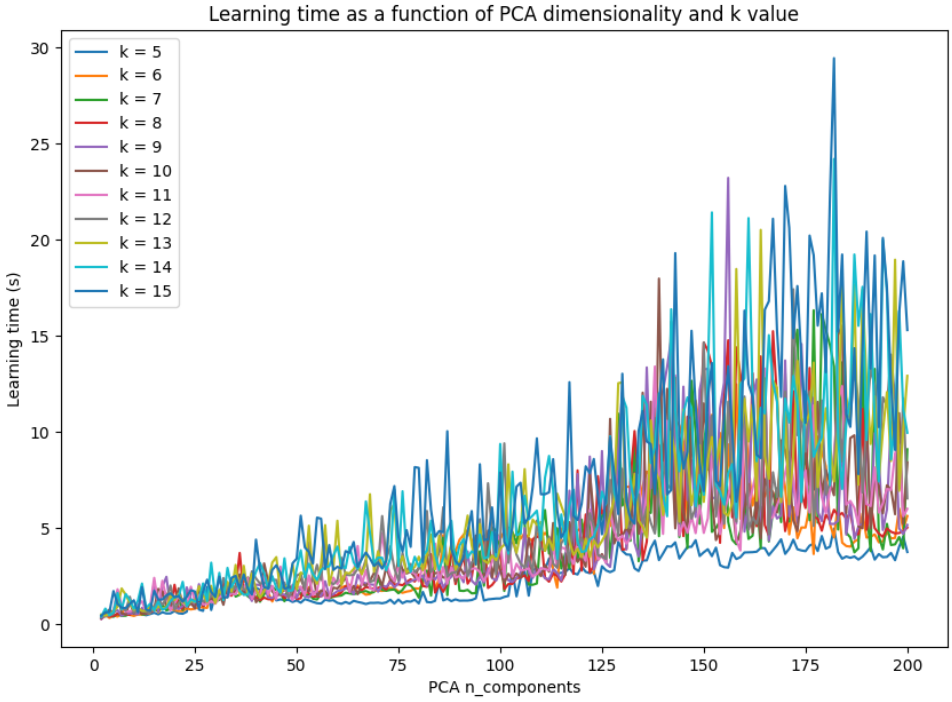
- ***Learning Time***: time needed for the algorithm to **complete the clustering**.
- ***Rand Score***: measure of the **similarity between two clusterings**. It compares the clustering assignments of **pairs of samples** between the predicted clustering and the true labels (in our case, fortunately, available). The **Rand score ranges from 0 to 1**, with 1 indicating identical clusterings, and 0 indicating that the two clusterings are completely different[2].

[1]It obviously also depends on the algorithm chosen

[2]The Rand score is then calculated as the sum of the number of agreements (pairs assigned to the same cluster in both labelings) and disagreements (pairs assigned to different clusters in both

4.2.1 Learning Time

Figure 4.7: Learning Time for Gaussian Mixture

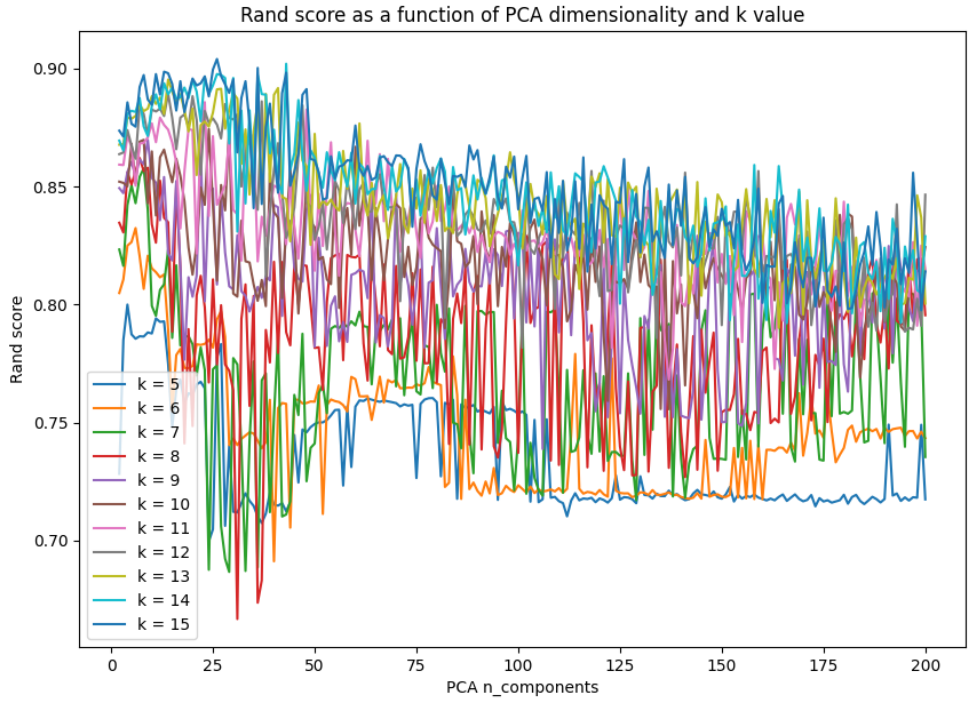


As shown in Figure 4.7, it is evident that both the **number of clusters** and the **PCA dimensionality** significantly **impact the learning time of our algorithm**. Increasing the PCA dimensionality results in a non-linear increase in time, with several fluctuations. On the other hand, the number of clusters has a more pronounced effect on the required time to cluster all vectors. It is noteworthy that the **performance of this algorithm is excellent**, with a maximum wait time of approximately 30 seconds in the worst-case scenario for clustering all items.

labelings), divided by the total number of pairs. So, it's possible to obtain very good results even if we are providing more clusters than the real number.

4.2.2 Rand Score

Figure 4.8: Rand Score for Gaussian Mixture



A very peculiar result is apparent here. Despite having more information with PCA of dimension 200, the **best result obtained is around 30**. Another noteworthy outcome is that a *better Rand score is obtained using 15 clusters* than the correct number of 10. Let us provide an interpretation of these results and attempt to offer some explanations:

- The reason why 200 is not the best PCA dimension can be explained by the fact that the **additional information does not significantly contribute to improving the clustering performance**. 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.

- The reason why better results are obtained by increasing the number of clusters can be attributed to the fact that **more clusters provide finer-grained distinctions between samples and can better capture the underlying structure of the data**. However, there is a trade-off between the number of clusters and the interpretability of the results, as a large number of clusters can make it difficult to interpret the meaning of each cluster. Furthermore, it is important to note that the **Rand score evaluates only if two items are in the same cluster** and does not provide a complete measure of the correctness of clustering.

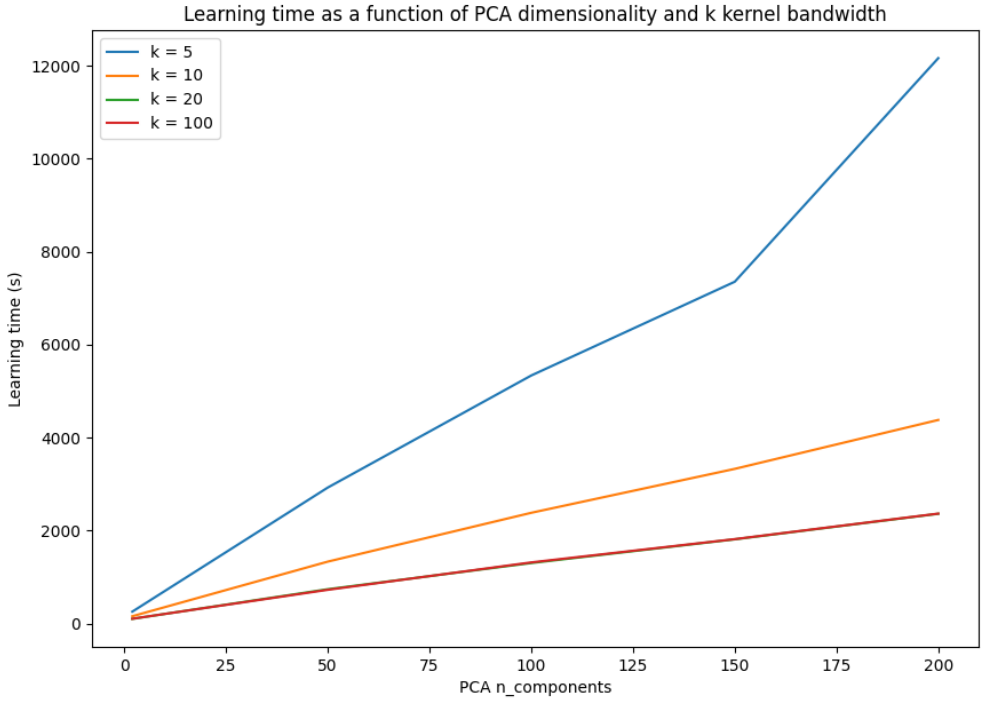
The ***best rand score*** is: 0.9040897185461738 obtained with `PCA n_components`: 26 and `k`: 15

4.3 Mean Shift

For the mean shift algorithm the parameters has been varied from PCA from 2 to 200 with steps of 25, while for ***kernel bandwidth*** we use the values: 5, 10 ,20 ,100.

4.3.1 Learning Time

Figure 4.9: Learning Time for Mean Shift



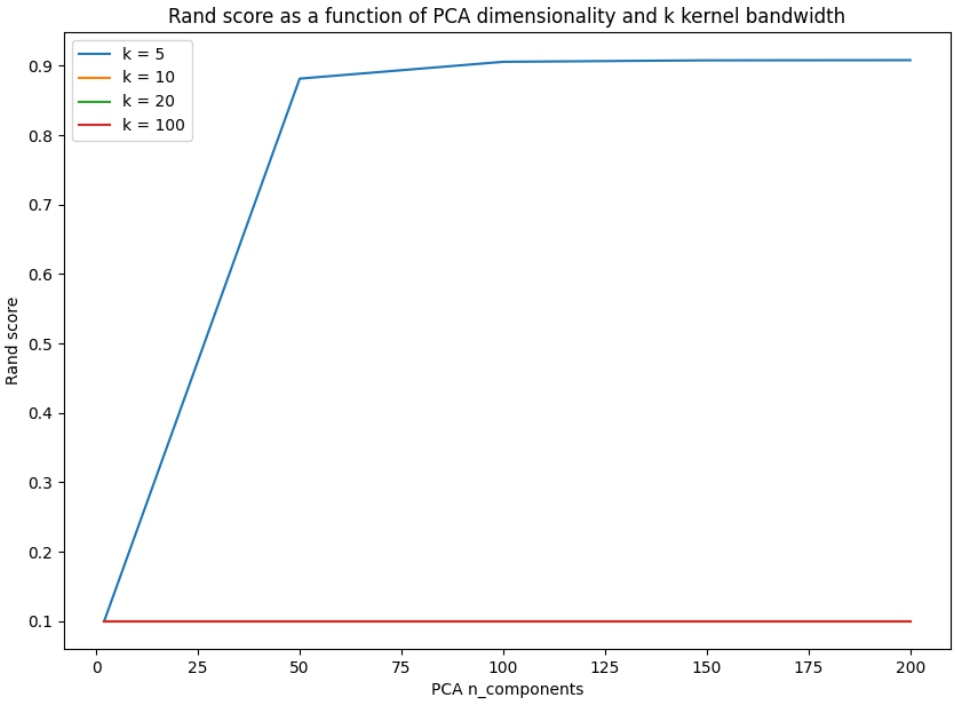
The learning time of the Mean Shift algorithm is **strongly related to the dimensionality of the input data**, which is determined by the number of PCA dimensions. This is because the Mean Shift algorithm needs to compute the pairwise distance between all points in the dataset, and as the number of dimensions increases, the **computational cost of calculating pairwise distances** also increases.

Furthermore, the bandwidth parameter used in the Mean Shift algorithm has a significant impact on the learning time. The bandwidth parameter determines the **size of the window** used to estimate the density of the data. If the bandwidth is set too small, the **algorithm may converge too slowly** or get stuck in local optima. (On the other hand, if the bandwidth is too large, the algorithm may converge too quickly or over-cluster the data.)

In the figure 4.9 we can see that when using a bandwidth of 5 with the Mean Shift algorithm, the learning time approaches almost 12000 seconds due to the high dimensionality of the data and the large number of pairwise distances that need to be calculated. Therefore, it is **very important to carefully select the bandwidth parameter**.

4.3.2 Rand Score

Figure 4.10: Rand Score for Mean Shift



When the **bandwidth is too large**, the algorithm will **group together points that are far apart**, leading to a low Rand score because the resulting clustering is very different from the true clustering. On the other hand, when the bandwidth is **too small**, the algorithm will identify many **small clusters that do not correspond to the true clusters**, leading again to a low Rand score. In this case, the Rand score is almost equal to 0 when the bandwidth is 10, 20, or 100 because these values are too large for the dataset, leading to over-merging of clusters. However, when the bandwidth is set to 5, the algorithm is able to identify the true clusters more accurately, resulting in a higher Rand score of 0.9.

Overall, mean shift clustering is a powerful algorithm that can be used to identify complex clusters and can handle a wide range of data types and shapes, but it can be **computationally expensive for large datasets**.

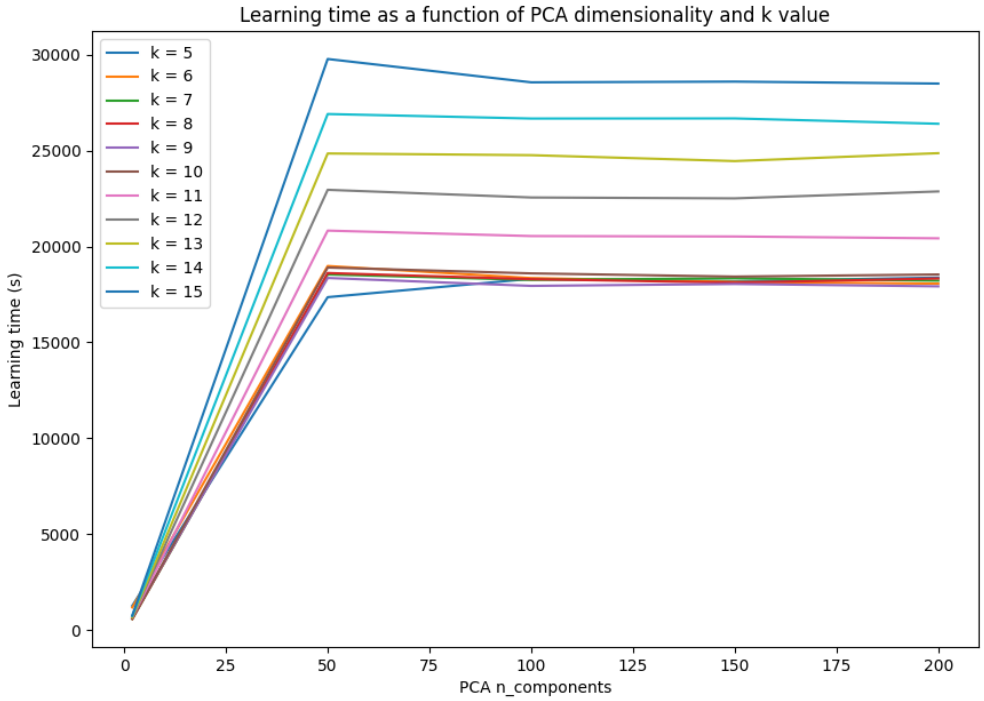
The ***best rand score*** is: 0.9080764847717147 obtained with `PCA n_components`: 200 and `k`: 5

4.4 Spectral clustering

For the spectral clustering algorithm, the parameters has been varied from PCA from 2 to 200 with steps of 50 (due to the computational requirements), while varying the number of clusters from 5 to 15.

4.4.1 Learning Time

Figure 4.11: Learning Time for Spectral clustering

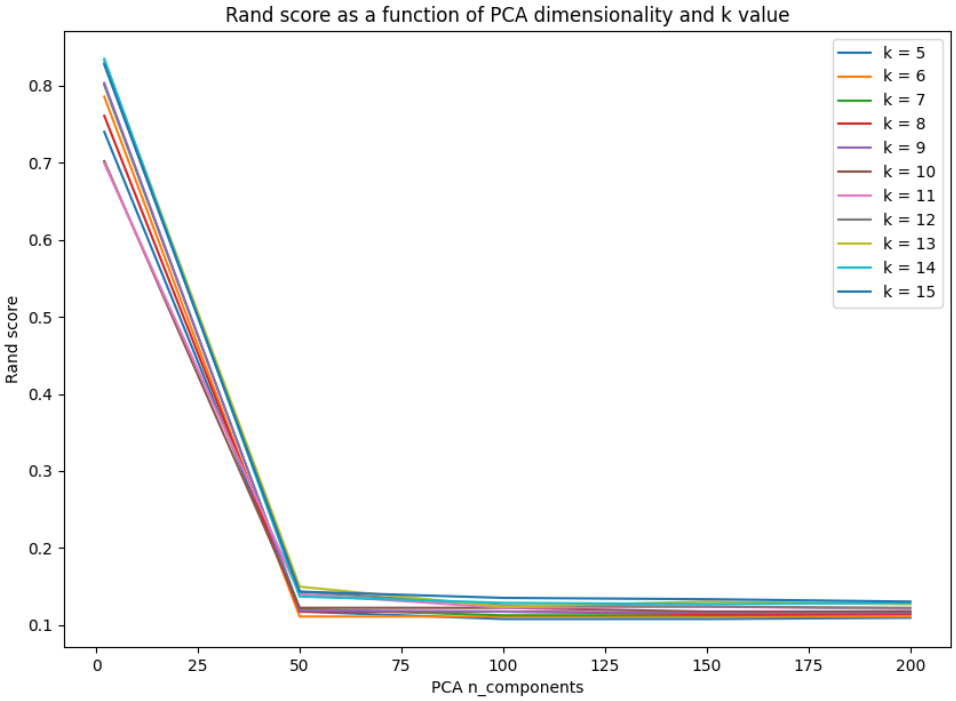


The learning time of the Spectral clustering algorithm is **strongly related to the dimensionality of the input data**, which is determined by the number of PCA dimensions. This is because the Spectral Clustering algorithm involves computing the eigenvalues and eigenvectors of a matrix derived from the input data, which can be computationally expensive as the dimensionality of the data increases.

Furthermore, the **number of clusters k also impacts the performance**, as the algorithm needs to perform k-means clustering on the eigenvectors. Therefore, as the dimensionality of the data and the number of clusters increase, the learning time of Spectral Clustering can become enormous. In fact, the maximum measured learning time for Spectral Clustering was *29773.807464132988* seconds, which is more or less 8 hours!

4.4.2 Rand Score

Figure 4.12: Rand Score for Spectral clustering



Recalling that the rand score is a measure of similarity between two sets of data and ranges from 0 to 1, with 1 indicating perfect similarity and 0 indicating no similarity, we immediately notice that the only useful results are obtained trough PCA with 2-dimension. It is possible that, for all the other dimensionality, the **algorithm was not able to finish its computation steps** due to insufficient RAM, leading to

meaningless results. This is a common issue when working with large data sets and computationally intensive algorithms.

In the current situation, even if the results are meaningless, it may be better to **leave them as they are** and acknowledge the limitations of the algorithm in order to compare these results with those obtained using the other clustering methods on a PC that has not the top hardware resources. This will help to determine the effectiveness of the algorithm in relation to other methods under similar conditions. The ***best rand score*** is: 0.83427628 obtained with `PCA n_components`: 2 and `k`: 14.

Chapter 5

Conclusion

In conclusion, we compare all the results obtained through the following table:

	Best Rand Score	**Learning Time**	**PCA**	**k**
Mixture of Gaussian	0.90408972	0.81881404s	26	15
Mean Shift	0.90807648	12161.4828136s	200	5
Spectral clustering	0.83427628	750.40353094	2	14

The Mean Shift clustering algorithm attained the highest Rand Score of 0.908, closely followed by the Mixture of Gaussian algorithm with a score of 0.904. However, while the Mixture of Gaussian algorithm had the shortest learning time of just 0.291 seconds, the Mean Shift algorithm took the longest, at 12161.483 seconds. In contrast, the Spectral clustering algorithm scored the lowest Rand Score of 0.834.

Regarding parameter settings, the Mean Shift algorithm required a large number of PCA components (200) and a small k value (5), while the Mixture of Gaussian algorithm needed a large k value (15) and a high number of PCA components (26).

Overall, for this specific evaluation, the Mixture of Gaussian algorithm seems to be the preferred choice since it achieved excellent results in less than a second! However, it's worth noting that these results apply only to this particular problem and were computed using a machine with limited performance. It's possible that the Spectral clustering algorithm could have achieved better results with a more powerful PC capable of handling higher PCA components.

	Best Learning Time	**Worst Learning Time**
Mixture of Gaussian	0.29094958305358887s	29.45444965362549s
Mean Shift	100.23182702064514s	12161.482813596725s
Spectral clustering	581.5151714712673	29773.807464132988s

Finally, let's compare the fastest learning time of each algorithm. It's evident that the Mixture of Gaussian algorithm stands out due to its remarkably fast computations. Conversely, the Spectral clustering algorithm, although highly effective and capable of achieving optimal results, does not perform as well on PCs with limited computational power.

In conclusion, it's essential to recognize that these results are specific to this particular problem and were obtained using a particular PC setup. Consequently, we cannot conclude that the Mixture of Gaussian algorithm is always the best clustering algorithm or that the Spectral clustering algorithm is the worst. However, what we can learn from these findings is that when we require an algorithm that performs well and needs to be trained in a limited environment, we should consider using the Mixture of Gaussians algorithm.